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
cat /etc/*release*	Check vendor release version
uname -r	Check Linux kernel version
nproc	Check # of CPUs
less /proc/cpuinfo	CPU specs
lscpu	Less verbose CPU specs
cat /proc/meminfo	Detailed memory usage
free -m	Check memory utilizations
df -m	Check disk utilizations
ifconfig	Basic statistics about network interfaces
netstat	Check port status. Add –ano | grep “<port”> to check a particular port
ulimit -a	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 `top` to get a quick assessment of whether the system is lightly or heavily loaded. The important columns include *pid*, *RES*, *R|S*, *%CPU*, *%MEM*, *TIME+*, and so on. Keep in mind that by default, `top` sorts by %CPU, but you can toggle sorting with Shift+M for %MEM, Shift+T for TIME+, and Shift+P for %CPU. Shift+I also allows you to turn Irix mode off so that the %CPU column would display total average rather than cumulative CPU utilizations.
CPU or IO bottlenecks	Run the `vmstat -n <interval> <count> -S M` command and: ■ If the **r** column is high, it means CPU is the bottleneck, as **r** means the # of processes in the run queue, waiting for a free CPU slot. ■ If the **b** column is high, it means IO (disk or network) is the bottleneck, as **b** means the # of processes waiting for a resource other than a CPU.
Hot processes	Run `top -c` to identify hot processes that have high %CPU, high %MEM, and high TIME+. Use *sort-by* as described above to toggle the top processes. To learn how a (hot) process is launched, run the `ps -fwwp <pid>` command, where `<pid>` can be found with the `top -c` command. You can also run the `top -d <interval> -b -n <count> -p <pid1>,<pid2>,...`command to obtain samples for the identified hot processes. Use the `pstree` command to find out the ancestor processes of a hot process, all the way to `init` or `systemd`.

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 cswch/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.

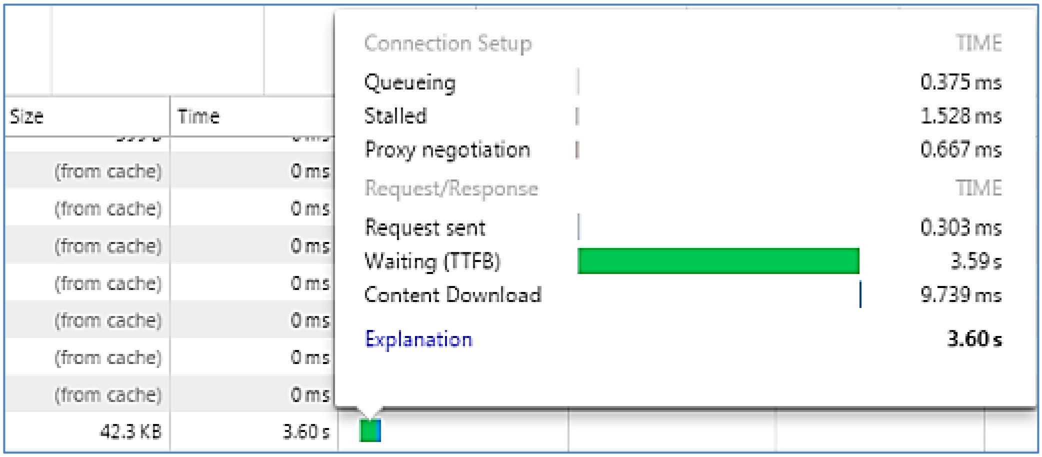
My Harp tool is a Java-based tool developed for offline post-analyzing HTTP traffic captured with Chrome Dev Tool (CDT) or FireBug with FireFox. Based on the HAR files saved with the CDT or FireBug, it parses the given input file in Har format, calculates some important metrics such as the server time, 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.



Size	Time
(from cache)	0 ms
(from cache)	0 ms
(from cache)	0 ms
(from cache)	0 ms
(from cache)	0 ms
(from cache)	0 ms
(from cache)	0 ms
42.3 KB	3.60 s

Connection Setup	TIME
Queueing	0.375 ms
Stalled	1.528 ms
Proxy negotiation	0.667 ms
Request/Response	TIME
Request sent	0.303 ms
Waiting (TTFB)	3.59 s
Content Download	9.739 ms
Explanation	**3.60 s**

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

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

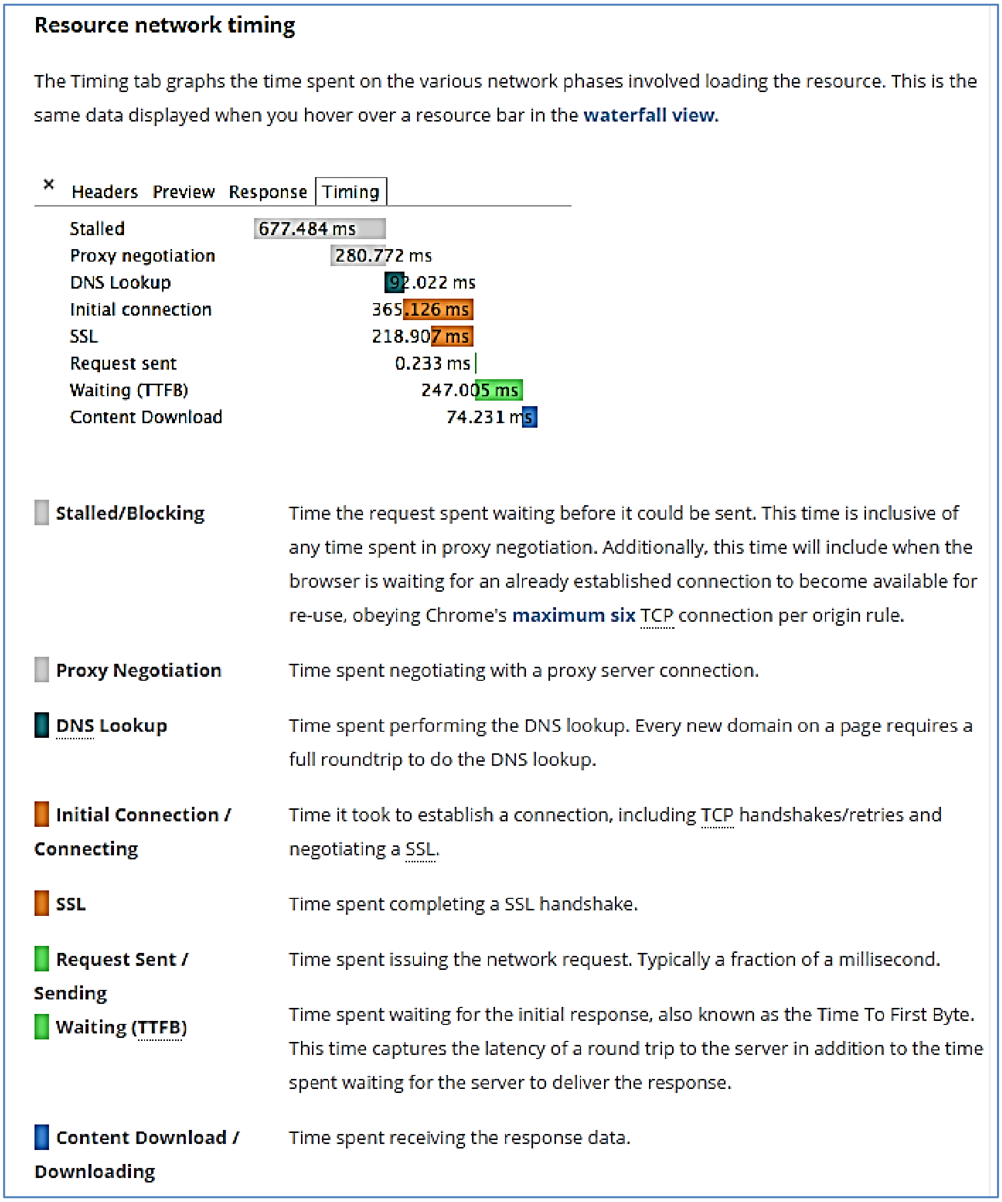
Steps to get started with using Harp:

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.



Resource network timing

The Timing tab graphs the time spent on the various network phases involved loading the resource. This is the same data displayed when you hover over a resource bar in the **waterfall view.**

Stalled	677.484 ms
Proxy negotiation	280.772 ms
DNS Lookup	92.022 ms
Initial connection	365.126 ms
SSL	218.907 ms
Request sent	0.233 ms
Waiting (TTFB)	247.005 ms
Content Download	74.231 ms

⬜ **Stalled/Blocking** Time the request spent waiting before it could be sent. This time is inclusive of any time spent in proxy negotiation. Additionally, this time will include when the browser is waiting for an already established connection to become available for re-use, obeying Chrome's **maximum six** TCP connection per origin rule.

⬜ **Proxy Negotiation** Time spent negotiating with a proxy server connection.

🟦 **DNS Lookup** Time spent performing the DNS lookup. Every new domain on a page requires a full roundtrip to do the DNS lookup.

🟧 **Initial Connection / Connecting** Time it took to establish a connection, including TCP handshakes/retries and negotiating a SSL.

🟧 **SSL** Time spent completing a SSL handshake.

🟩 **Request Sent / Sending** Time spent issuing the network request. Typically a fraction of a millisecond.

🟩 **Waiting (TTFB)** Time spent waiting for the initial response, also known as the Time To First Byte. This time captures the latency of a round trip to the server in addition to the time spent waiting for the server to deliver the response.

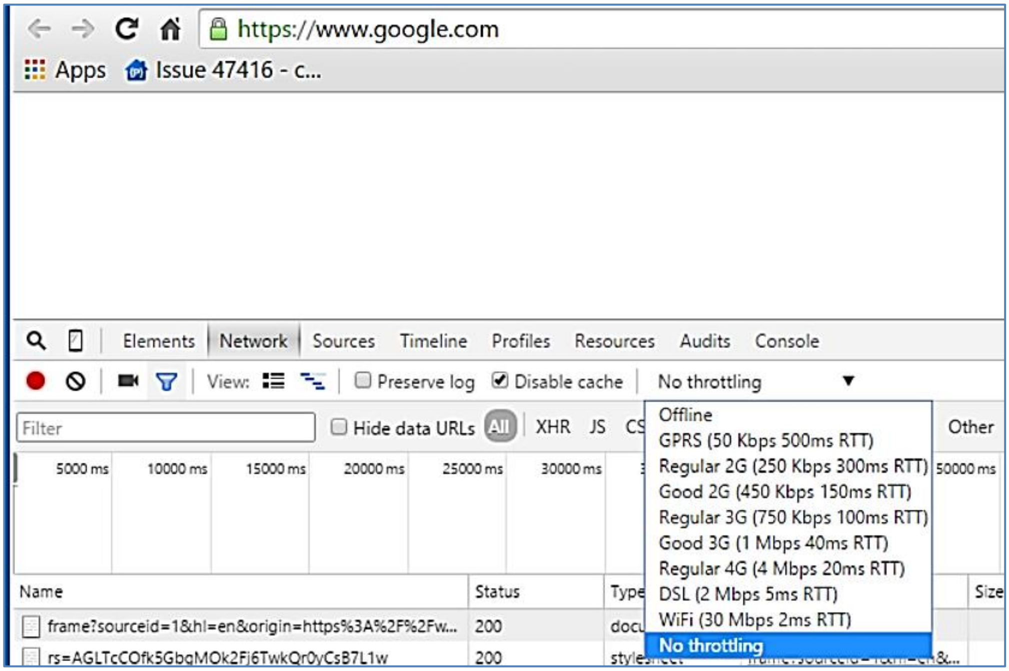
🟦 **Content Download / Downloading** Time spent receiving the response data.

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.

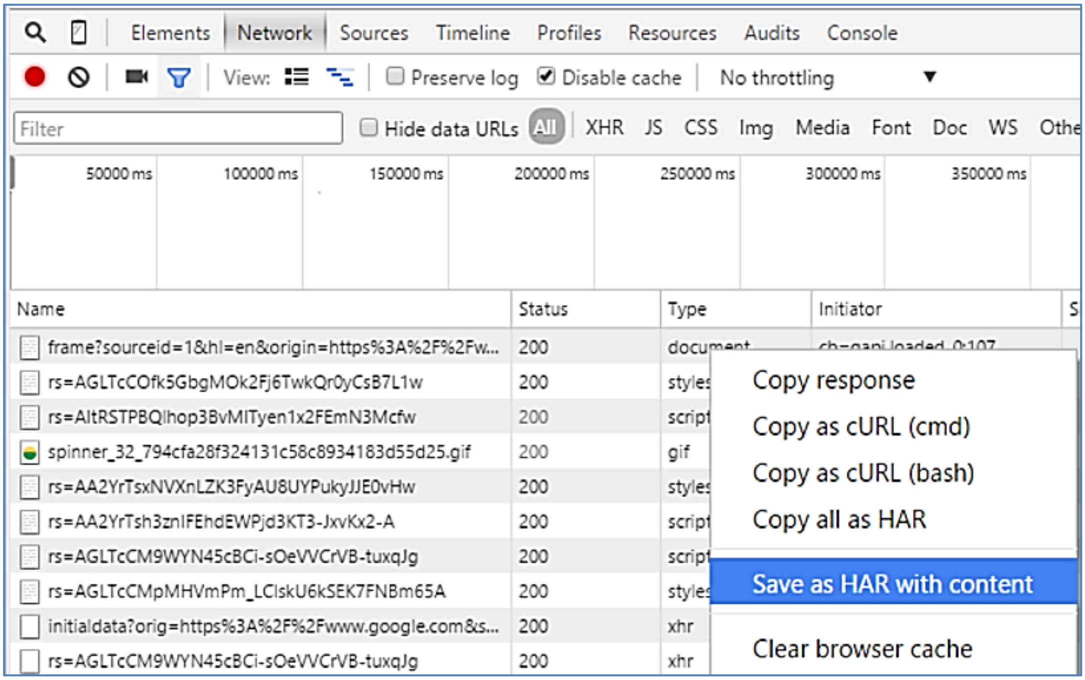


□ Preserve log ☑ Disable cache No throttling ▼

□ Hide data URLs

Name	Status	Type	Size
frame?sourceid=1&hl=en&origin=https%3A%2F%2Fw...	200	docu	
rs=AGLTcCOfk5GbqMOk2Fj6TwkQr0yCsB7L1w	200	style	

Figure F.3 CDT with Chrome



☐ Preserve log ☑ Disable cache No throttling ▼

☐ Hide data URLs

Name	Status	Type	Initiator	S
frame?sourceid=1&hl=en&origin=https%3A%2F%2Fw...	200	document	cb=gapi.loaded_0:107	
rs=AGLTcCOfk5GbgMOk2Fj6TwkQr0yCsB7L1w	200	styles		
rs=AItRSTPBQlhop3BvMITyen1x2FEmN3Mcfw	200	script		
spinner_32_794cfa28f324131c58c8934183d55d25.gif	200	gif		
rs=AA2YrTsxNVXnLZK3FyAU8UYPukyJJE0vHw	200	styles		
rs=AA2YrTsh3znlFEhdEWPjd3KT3-JxvKx2-A	200	script		
rs=AGLTcCM9WYN45cBCi-sOeVVCrVB-tuxqJg	200	script		
rs=AGLTcCMpMHVmPm_LClskU6kSEK7FNBm65A	200	styles		
initialdata?orig=https%3A%2F%2Fwww.google.com&s...	200	xhr		
rs=AGLTcCM9WYN45cBCi-sOeVVCrVB-tuxqJg	200	xhr		

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.

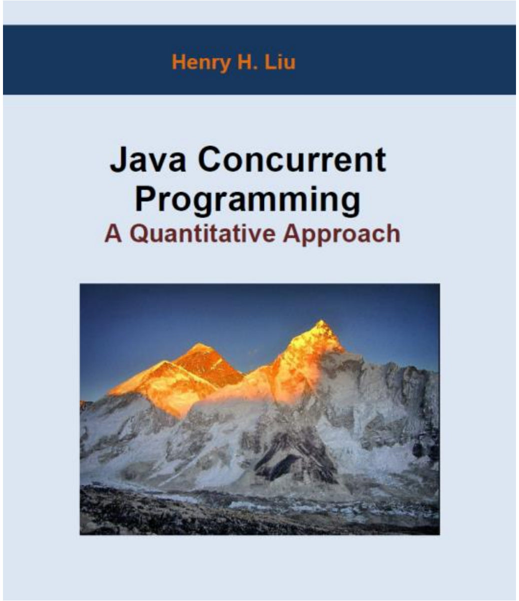
5 If you see gaps between adjacent requests, most likely, caching is not implemented properly on the server side. With one of my projects, large gaps were observed with some user actions, and those large gaps disappeared after extension IDs of plug-ins on the server side were cached properly. You should 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

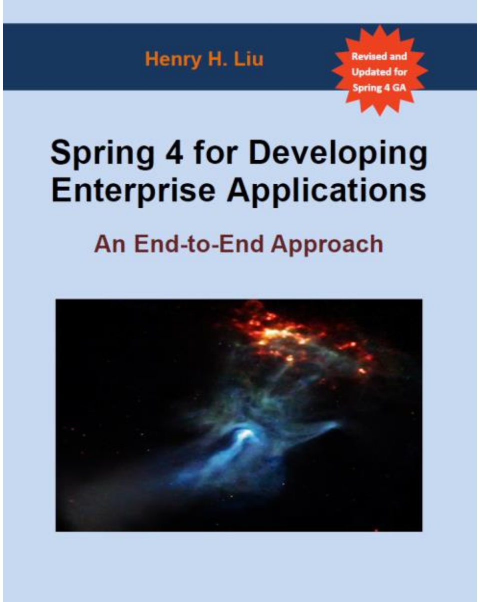
Other texts by the same author, available on Amazon:



Henry H. Liu

Java Concurrent Programming

A Quantitative Approach



Henry H. Liu

Revised and Updated for Spring 4 GA

Spring 4 for Developing Enterprise Applications

An End-to-End Approach