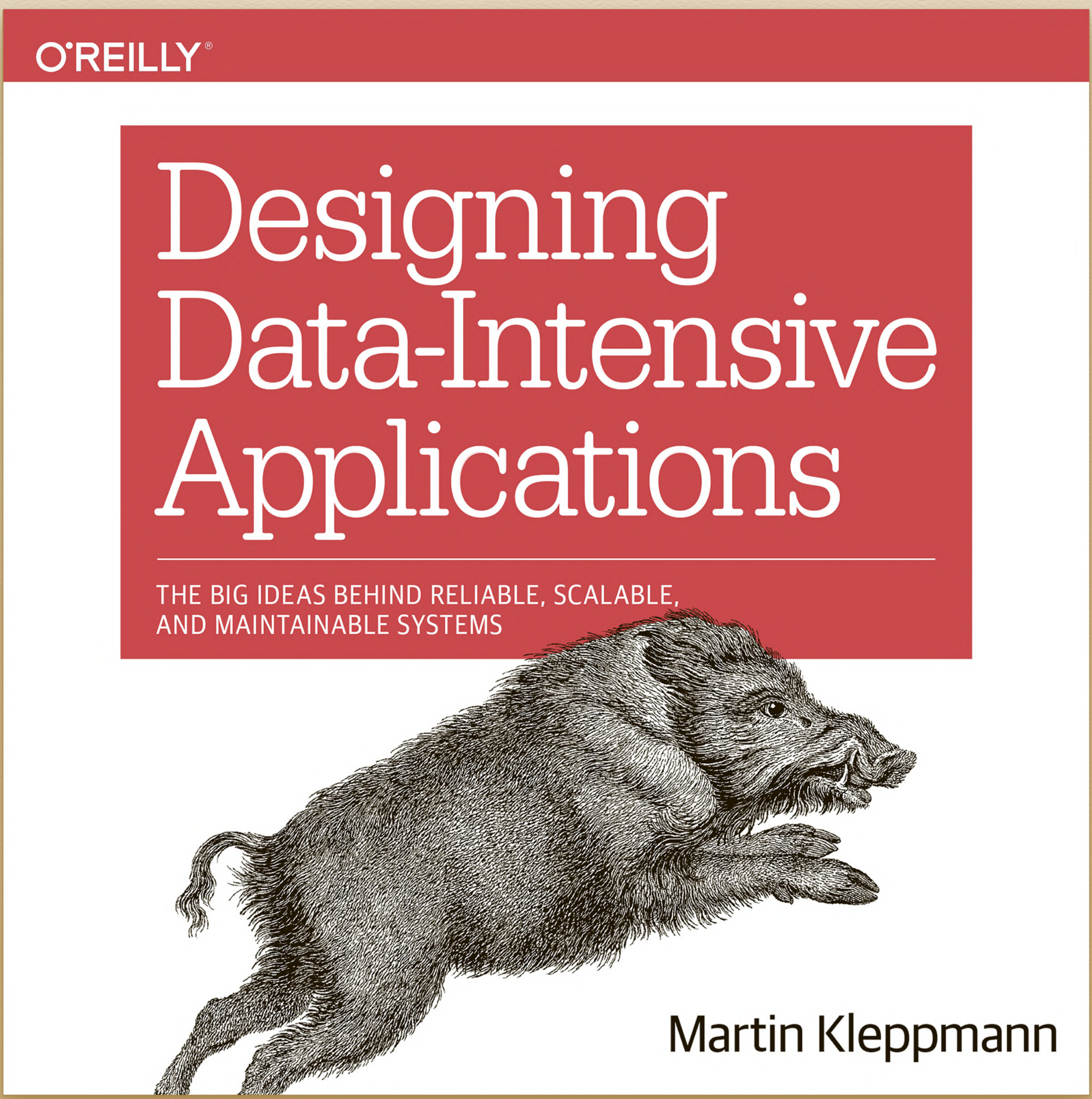

O'REILLY®

Designing Data-Intensive Applications

THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS

Martin Kleppmann

Upfront Books

FIGURE 10-1

CHAIN OF UNIX COMMANDS

```
cat /var/log/nginx/access.log | ❶
  awk '{print $7}' | ❷
  sort             | ❸
  uniq -c          | ❹
  sort -r -n       | ❺
  head -n 5          ❻
```

❶ Read the log file.

❷ Split each line into fields by whitespace, and output only the seventh such field from each line, which happens to be the requested URL. In our example line, this request URL is */css/typography.css*.

❸ Alphabetically `sort` the list of requested URLs. If some URL has been requested *n* times, then after sorting, the file contains the same URL repeated *n* times in a row.

❹ The `uniq` command filters out repeated lines in its input by checking whether two adjacent lines are the same. The `-c` option tells it to also output a counter: for every distinct URL, it reports how many times that URL appeared in the input.

❺ The second `sort` sorts by the number (`-n`) at the start of each line, which is the number of times the URL was requested. It then returns the results in reverse (`-r`) order, i.e. with the largest number first.

❻ Finally, `head` outputs just the first five lines (`-n 5`) of input, and discards the rest.

The output of that series of commands looks something like this:

```
4189 /favicon.ico
3631 /2013/05/24/improving-security-of-ssh-private-keys.html
2124 /2012/12/05/schema-evolution-in-avro-protocol-buffers-thrift.html
1369 /
 915 /css/typography.css
```

Although the preceding command line likely looks a bit obscure if you're unfamiliar with Unix tools, it is incredibly powerful. It will process gigabytes of log files in a matter of seconds, and you can easily modify the analysis to suit your needs. For example, if you want to omit CSS files from the report, change the `awk` argument to `'$7 !~ /\.css$/ {print $7}'`. If you want to count top client IP addresses instead of top pages, change the `awk` argument to `'{print $1}'`. And so on.

FIGURE 10-2

RUBY PROGRAM

```
counts = Hash.new(0) ❶

File.open('/var/log/nginx/access.log') do |file|
  file.each do |line|
    url = line.split[6] ❷
    counts[url] += 1 ❸
  end
end

top5 = counts.map{|url, count| [count, url] }.sort.reverse[0...5] ❹
top5.each{|count, url| puts "#{count} #{url}" } ❺
```

❶ `counts` is a hash table that keeps a counter for the number of times we've seen each URL. A counter is zero by default.

❷ From each line of the log, we take the URL to be the seventh whitespace-separated field (the array index here is 6 because Ruby's arrays are zero-indexed).

❸ Increment the counter for the URL in the current line of the log.

❹ Sort the hash table contents by counter value (descending), and take the top five entries.

❺ Print out those top five entries.