R Packages to Aid in Handling Web Access Logs

by Oliver Keyes, Bob Rudis

Abstract Web access logs contain information on HTTP(S) requests and form a key part of both industry and academic explorations of human behaviour on the internet - explorations commonly performed in R. In this paper we explain and demonstrate a series of packages - **webreadr**, **urltools**, **iptools** and **rgeolocate** - designed to efficiently read in, parse and munge access log data, allowing researchers to handle it easily.

Introduction

The rise of the World Wide Web over the last few decades has made it dramatically easier to access and transfer data, and R (R Core Team, 2015) boasts abundant functionality when it comes to taking data *from* the web. Base R itself has simple file downloading and page reading capabilities, through the `download.file` and `readLines` functions, and additional functionality is made available for handling web-accessible data through packages such as **httr** (Wickham, 2015).

Data *on* the web is not, however, the only kind of web data that interests researchers; web traffic is, in and of itself, an interesting data source. Access logs - records of connections between users and a web server - are an asset and resource for people studying everything from user behaviour on the internet (Halfaker et al., 2014), to website performance (Ryckbosch and Diwan, 2014), to information security (Bhingarkar and Shah, 2015).

As a statistically-oriented programming language, R is commonly used by these same researchers for data analysis, testing and reporting - but it lacks tools designed for the kinds of data sources and formats that are encountered when dealing with access logs. In this article we review the use cases for particular operations over web data, the limitations in base R when it comes to performing those operations, and a suite of R packages designed to overcome them: **webreadr** (Keyes, 2015), designed for reading access logs in, **urltools** (Keyes et al., 2015a), for URL manipulation, **iptools** (Keyes et al., 2015b) for handling IP addresses, and **rgeolocate** (Keyes et al., 2015b) for direct IP geolocation.

Reading access logs

The first task with any data analysis is to read the data into R so that it can be manipulated. With access logs this is slightly complicated by the fact that there is no one standard for what a log should look like; instead, there are multiple competing approaches from different software platforms and eras. These include the Common Log Format (CLF), the confusingly-named Combined Log Format, and formats used by individual, commonly-used software platforms - such as the custom format for the Squid internet caching software, and the format used by Amazon Web Services (AWS).

The difference between formats can easily be shown by looking at how timestamps are represented:

Table 1: Timestamps in Common Access Log Formats

Log Type	Timestamp Columns	Timestamp Format
Common Log Format	1	10/Oct/2000:13:55:36 -0700
Combined Log Format	1	26/Apr/2000:00:23:48 -0400
Squid	1	1286536309.450
AWS	2	2014-05-23 01:13:11

With four log types, we have three different timestamp formats - and that's only one of the many columns that could appear. These logs also vary in whether they specify quoting fields (or sanitising unquoted ones), the columns they contain and the data each column contains in turn.

To make reading access logs into R as easy as possible we created the **webreadr** package. This contains user-friendly equivalents to `read.table` for each type of log, detecting the fields that should appear, converting the timestamps into POSIX objects, and merging fields or splitting fields where necessary. The package contains four core functions, one for each form of log: `read_clf` for the Common Log Format, `read_combined` for the Combined Log Format, and `read_squid` and `read_aws` for Squid and AWS formats respectively. Each one abstracts away the complexity of specifying column names and formats, and instead allows a researcher to read a file in with a minimal amount of work.

As the name suggests, it is built not on top of base R but on top of the **readr** (Wickham and Francois) package, allowing us to take advantage of substantial speed improvements that package's base functions have over base R. These improvements can be seen in the visualisation below, which uses **microbenchmark** (Mersmann, 2014) to compare 100 reads of a 600,000-line "squid" formatted file with webreadr to the same operation performed in base R:

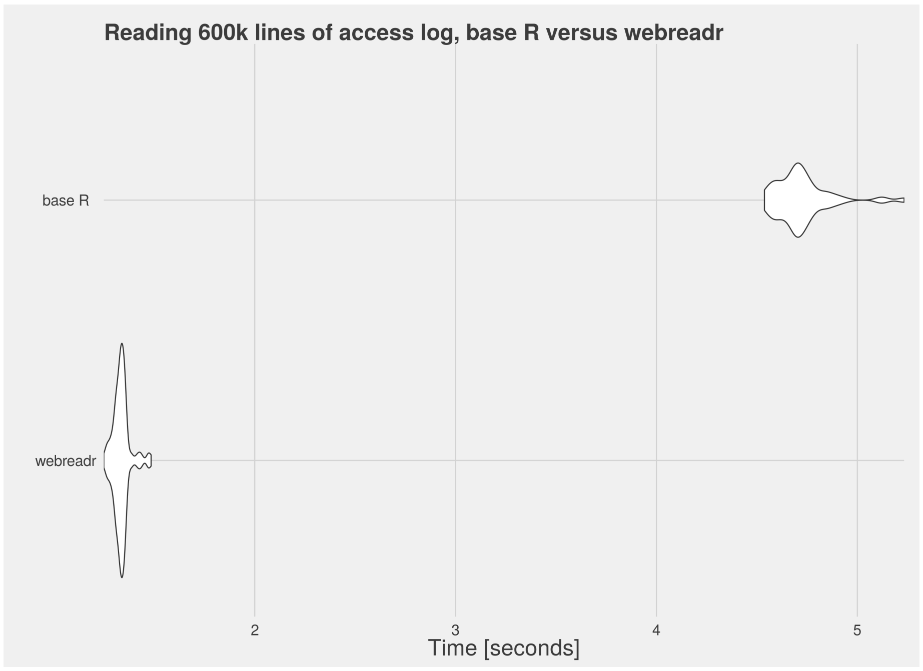

Figure 1: Results of microbenchmark run: read_squid versus base-R equivalent code

webreadr takes 1.2 seconds to read this file in at a minimum, and 1.48 at maximum, with a median of 1.33. This is consistently 3.5-6 times faster than the equivalent base R functionality - and is also, as explained, far easier to use.

Decoding and parsing URLs

Within access logs, URLs appear - both to describe the web asset or page that the user requested and where the user came from. These fields are usually named url and referer respectively.

Decoding

Both values can be percent-encoded, allowing them toinclude characters that are valid but reserved by the URL specification as having special meanings ("reserved characters"). A # symbol, for example, is encoded as %23: a percentage symbol, followed by the byte-code for that character.

The encoding of reserved characters is useful, since it means that URL paths and queries can contain a vast array of values - but it makes data analysis tougher to do. Examples of common data analysis or cleaning operations that become more difficult are:

1. **Aggregation**. Aggregating URLs together is useful to identify, for example, the relative usage and popularity of particular pages in your data - but it becomes tougher if encoding is inconsistent, because two URLs could hold the same value but *look* very different.
2. **Value selection**. With text-based data, regular expressions are a common way of filtering or selecting entries that meet particular conditions, but things become fuzzy when you have to look not just for particular characters (a space, say) but also the encoded values (%20).
3. **Exploratory data analysis** (EDA). EDA is a common initial step to investigate a data set, examining the variables and values it contains and whether they meet a researcher's expectations - but on a practical basis it becomes difficult when the values aren't human-readable.

The solution is to be able to consistently decode URLs, which makes URL-based data far easier to analyse. Base R does contain a function, URLdecode, for doing just that, but is neither vectorised nor based on compiled code; with large datasets it takes an extremely long time.

To solve this common problem in analysing request logs, the **urltools** [@urltools] package was created. This contains a function, url_decode, which decodes URLs - but relies on vectorised, compiled code. Benchmarking the two approaches against each other shows that the `urltools` implementation is approximately 60-70 times faster over large datasets. Again using **microbenchmark**, if we compare the vectorised decoding of 1,000,000 URLs with `urltools` against `URLdecode` in a `vapply` loop, we see a 60-70 times speed improvement of urltools over base R:

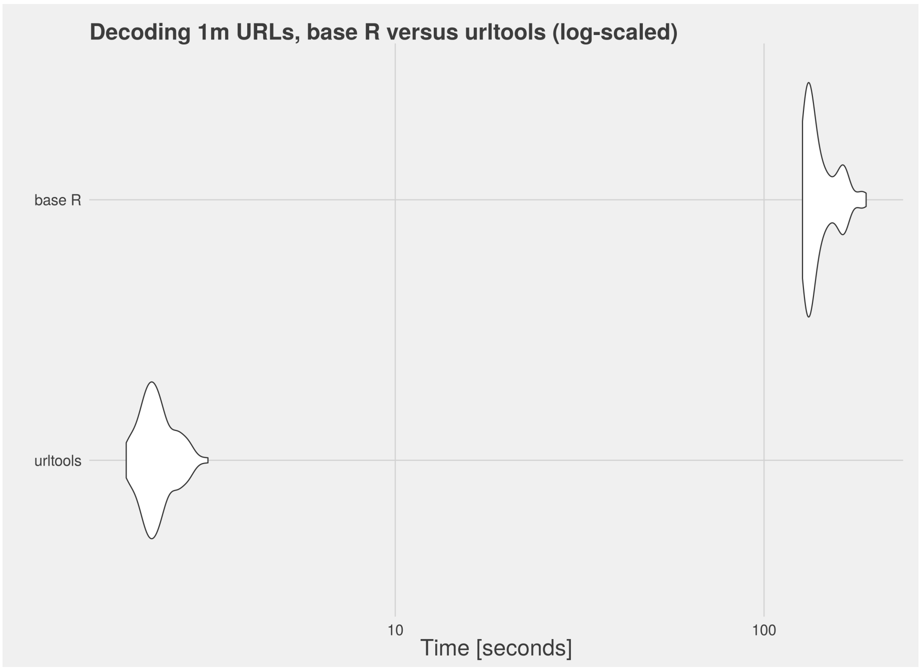

Figure 2: Results of microbenchmark run: url_decode versus base-R equivalent code

Parsing

The standard for URLs (Berners-Lee et al., 1994) divides them into a heirarchical sequence of components - the scheme ("http"), host ("en.wikipedia.org"), port ("800"), path ("wiki/Main_Page") and searchpart, or parameters ("action=edit"). Together, these make up a URL ("`http://en.wikipedia.org:800/wiki/Main_Page?action=edit`").

Parsing URLs to isolate and extract these components is a useful ability when it comes to exploring request logs; it lets a researcher pick out particular schemes, paths, hosts or other components to aggregate by, identifying how users are behaving and what they are visiting. It makes anonymising data - by removing, for example, the parameters or path, which can contain relatively unique information - easier.

Base R does not have native code to parse URLs, but the **httr** package (Wickham, 2015) contains a function, `parse_url`, designed to do just that and built on base R's capabilities. This function is neither vectorised nor based on compiled code, and produces a list, making looping over a set of URLs to parse each one a somewhat thorny experience. This is totally understandable given the intent behind that function, which is to decompose individual URLs within the context of making HTTP requests, rather than to analyse request logs *en masse*.

`urltools` contains `url_parse` - which does the same thing as the equivalent `httr` functionality, but in a vectorised way, relying on compiled code, and producing a data.frame. Within the context of parsing and processing access logs, this is far more useful, because it works efficiently over large sets: httr's functionality, not having designed to be, does not. Indeed, benchmarking showed that httr's version is (for large vectors, at least) approximately 570 times slower than `url_parse`:

Summary

This file is only a basic article template. For full details of *The R Journal* style and information on how to prepare your article for submission, see the Instructions for Authors.

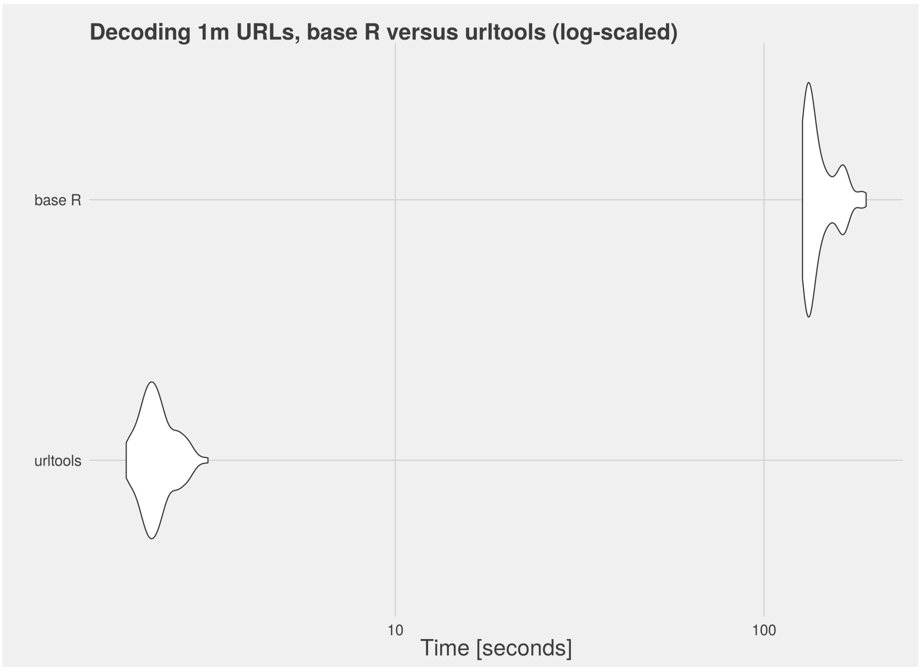

Figure 3: Results of microbenchmark run: url_parse versus httr equivalent code

Bibliography

T. Berners-Lee, L. Masinter, and M. McCahill. Uniform resource locators (url). RFC 1738, December 1994. URL `https://tools.ietf.org/html/rfc3986`. [p3]

A. S. Bhingarkar and B. D. Shah. A survey: Securing cloud infrastructure against edos attack. In *Proceedings of the International Conference on Grid Computing and Applications (GCA)*, page 16. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015. [p1]

A. Halfaker, O. Keyes, D. Kluver, J. Thebault-Spieker, T. T. Nguyen, K. Shores, A. Uduwage, and M. Warncke-Wang. User session identification based on strong regularities in inter-activity time. *CoRR*, abs/1411.2878, 2014. URL `http://arxiv.org/abs/1411.2878`. [p1]

O. Keyes. *webreadr: Tools for Reading Formatted Access Log Files*, 2015. URL `http://CRAN.R-project.org/package=webreadr`. R package version 0.3.0. [p1]

O. Keyes, J. Jacobs, M. Greenaway, and B. Rudis. *urltools: Vectorised Tools for URL Handling and Parsing*, 2015a. URL `http://CRAN.R-project.org/package=urltools`. R package version 1.3.2. [p1]

O. Keyes, D. Schmidt, D. Robinson, I. Maxmind, and P. Gloor. *rgeolocate: IP Address Geolocation*, 2015b. URL `http://CRAN.R-project.org/package=rgeolocate`. R package version 0.5.0. [p1]

O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2014. URL `http://CRAN.R-project.org/package=microbenchmark`. R package version 1.4-2. [p2]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. URL `https://www.R-project.org/`. [p1]

F. Ryckbosch and A. Diwan. Analyzing performance traces using temporal formulas. *Software: Practice and Experience*, 44(7):777–792, 2014. [p1]

H. Wickham. *httr: Tools for Working with URLs and HTTP*, 2015. URL `http://CRAN.R-project.org/package=httr`. R package version 1.0.0. [p1, 3]

H. Wickham and R. Francois. *readr: Read Tabular Data*. URL `https://github.com/hadley/readr`. R package version 0.1.1.9000. [p2]

Oliver Keyes
Wikimedia Foundation

line 1
line 2
`author1@work`

Bob Rudis
Rapid7
line 1
line 2
`author2@work`