ISC Proposal: An External R Sampling Profiler

Aaron Jacobs

2019-10-14

Signatories

Project team

- **Aaron Jacobs**, Data Scientist at Crescendo Technology

Consulted

- **Kirill Müller**, on technical feasibility and the relationship between this project and the **jointprof** package.
- **Jeroen Ooms**, on technical feasibility of a Windows port.

Motivation

How do R programmers identify and track down performance issues in their code? As usual, we can and should try to collect data to answer these questions.

Many R users will be familiar with using the built-in sampling profiler `Rprof()` to generate data on what their code is doing, and there are several excellent tools to facilitate understanding these samples (or serve as a front-end), including the **profvis** package.

However, the reach of `Rprof()` and related tools is limited: the profiler is "internal", in the sense that it must be manually switched on to work, either during interactive work (for example, to profile an individual function), or perhaps by modifying the script to include `Rprof()` calls before running it again.

At present, there is no way to collect useful information on R code that cannot by manually instrumented with `Rprof()` or that is already running. This makes it very difficult to answer questions like the following:

- How does an R application that you didn't write – and therefore have limited understanding of how to instrument – spend its time?
- An R program is misbehaving in production, taking far longer to run (or do its work) than we expected or experienced in other environments. What is it doing?

Instead, R programmers usually resort to trying to reproduce these problems in local, interactive sessions.

Prior Art

Many existing programming languages have one or more "external" profilers available, for example:

- The Linux system profiler `perf` will print sampled C/C++ stack traces (among many, many others).
- The `jstack` program for Java (among others).
- The Windows `VSPerfCmd` tool for C#/.NET applications.
- Pyflame for Python; and
- rbspy for Ruby.

All of these programs can "attach" (in one way or another) to a running process and inspect its memory at some frequency to understand what the current language-specific stack looks like. The last two are especially appealing, because they, like R, are interpreted languages, and show that it should be possible to produce a similar profiler for R.

Moreover, these external sampling profilers have proven extremely useful for diagnosing and fixing performance issues (or other bugs) in production environments. As R moves more and more into this space, I believe this could provide a substantial benefit to the community.

To our knowledge, there are no existing external sampling profilers for R, other than the demo discussed below.

The proposal

Overview

Project member Aaron Jacobs wrote a demo-quality external sampling profiler for R earlier this year, which is available on GitHub. The demo is also capable of outputting "mixed-mode" combinations of the R and C/C++ stacks, in line with the goals of the **jointprof** package.

This demo (1) proves that this is indeed possible for R; and (2) points the way to the technologies that are needed to make it work.

We propose to turn this demo-quality project into a robust set of tools that can be used by the wider community, and especially in **jointprof**.

A larger goal of this proposal is to help move the **jointprof** project forward. Provided that this project can meet its goals, the resulting library could replace the use of `pprof` in the **jointprof** package, giving that project three desirable features that it does not currently have:

- Support for profiling external R programs, which would unlock the ability to profile multiprocess-style parallel programs.
- Better profiling in the presence of R's lazy evaluation; and
- The possibility of support for Windows.

In addition, it could potentially remove the need for end users to install and manage a Go toolchain.

Detail

In order to make the existing demo more widely useful, we propose to break it into two components: a command-line tool for users and a C library that backs this program and other interfaces. This will involve:

- Refactoring the existing code to be more robust, handle more diverse R environments, and report errors to the caller; and
- Be sufficiently portable to work on any Linux distribution and any sufficiently recent version of R.

It is our view that a tool for R users is unlikely to be successful if it does not have an R-level interface. To that end, the second major goal of this project would be to create an R package backed by the same library. Provided that this library is relatively portable and written in C, we believe that such a package would be admissible to CRAN.

The delivery of

1. (i) a C library, (ii) a command-line program; and (iii) an R package for Linux

is **the minimum viable product (MVP) of this proposal**.

In addition to this MVP, the secondary goal of this proposal is

2. To add sampling of C/C++ stacks to this library, incorporating the lessons learned from the **jointprof** package.

Sampling of C/C++ stacks is already part of the existing demo (see the `libunwind` branch); however, collating these and the R-level stacks does not yet work reliably. In addition, we are unsure of whether the existing approach can be made to work on non-Linux platforms; Jeroen Ooms has provided some guidance on this question and possible alternatives.

Finally, we propose to "port" the tool so that it works in two other common R environments:

3. Under Docker. Although Docker is, of course, Linux, its use of process namespaces render the existing demo nonfunctional. Yet Dockerized R is very likely the medium-term future of production in R. The Pyflame profiler for Python has shown that it is possible to circumvent the namespace issues; we feel we could make the same approach work for this project.
4. Under Windows. Although there are likely fewer "production" deployments of R on Windows, it is a major platform for R users. The rbspy profiler for Ruby has shown that it is possible to support Windows in this way; we feel we could learn from them and make the same approach work for this project.

Priorities

The authors of this proposal are open to comments from the reviewers on what features ought to be prioritized.

Alternatives

A new, external sampling profiler is not the only way to achieve live, opt-in profiling of running R programs. It would also be possible to modify R itself to switch on profiling in response to some kind of IPC, such as a Unix signal. The authors have not raised this suggestion to the R developers, but it is still possible to do so.

However, there are major drawbacks to such an undertaking: it would require modifying R itself, perhaps in ways that have an undesirable impact on its complexity or performance, and would only allow code running under new versions of R (perhaps quite some time from now) to be profiled in this way. As a result, we believe a new tool is a better way to accomplish the project's goals.

Project plan

Start-up phase

The existing code is already on GitHub, and that platform is intended to be where all development takes place. Since the goals of this proposal are relatively well-defined, GitHub issues can be used to track progress and provide public visibility of progress. Moreover, the use of GitHub will give us access to tools such as Travis-CI, which can be used to ensure that the code will work under a variety of environments.

In addition, we have corresponded with Kirill Müller on the relationship between the proposed project and other packages and tools in the R ecosystem. Because of this, we believe that our views of future success are aligned with other projects and proposals, and that there are a few points of external reference against which to gauge the relevance of design or technical decisions.

Licensing

Since the demo already contains portions of the source code of R itself and this is unlikely to change in its transition to a library, it is very likely it must fall under the same license, GPLv2. This is a common license in the R community, so this choice is not expected to cause any friction.

Technical delivery

The timeline of the project is intended to be short:

1. Delivery of the proposed MVP: refactoring of the demo into a portable C program and library for Linux, plus an R package interface. Estimated to take up to two weeks. This would be signified by a "0.1" release of these artifacts.
2. Delivery of the proposed port to Docker. Estimated to take a few days.
3. Delivery of the proposed port to Windows of the R sampling portion of the library. Estimated to take up to two weeks.
4. Delivery of support for mixed-mode R and C/C++ stack sampling, again Linux-first. Estimated to take up to a month, and would be signified by a second release of the artifacts.
5. Delivery of wrap-up work, including a substantive blog post, integration into **jointprof**, and submission of one or more packages to CRAN. Estimated to take up to two weeks.

Given the plan to work nearly full-time on this proposal, we believe this should be possible.

Other aspects

In order to publicise the project and ensure that R users can actively benefit from it, we propose writing a blog post, ideally for the R Consortium, explaining how to use the tools provided by the project.

Since the goals of the project are relatively narrow and use cases are fairly narrow, we believe a single tutorial/evangelist-style post will be the most appropriate reference point for the community.

In addition, the authors of this proposal intend to publicize this work at all available opportunity: by submitting talks to UseR!, producing blog posts, demos, use cases, and possibly screencasts to be shared on Twitter and other platforms.

Requirements

People

The large majority of the work is expected to be carried out by project member Aaron Jacobs. Kirill Müller has volunteered to help oversee integrating the result of this work into the **jointprof** package.

Processes

We propose that the natural home of this project for future contributors is the `r-prof` organization, which is the umbrella group that currently oversees the **profile** and **jointprof** packages.

Tools & Tech

No special tools, tech, or equipment beyond the existing resources of project members will be required.

Funding

We request a total of $8,500 to support the equivalent of 1.5 months salary for project member Aaron Jacobs.

Summary

The main requirement for this project is funding to cover development costs, as all of the deliverables of this proposal are code and software artifacts.

Project member Aaron Jacobs has arranged with his employer to spend up to 80 percent of his time over the course of eight weeks on unpaid leave to work on this project, should it receive funding. His employer will continue to permit access to existing compute resources to him during this time.

The award size in this case is similar to those of the successful sftraj and Licensing R proposals, namely that it essentially covers the cost of salary, and over a similar period.

Success

Definition of done

Success can be defined as full delivery of the artifacts and outcomes discussed in the Technical delivery section above. Partial success can also be defined in this manner as delivery of some but not all these artifacts and outcomes.

Measuring success

As mentioned in the Technical delivery section, we can use the following markers:

- Release of version 0.1 of the proposed MVP library, command-line program, and R package.
- Release of the MVP artifacts with support for Docker.
- Release of the MVP artifacts with support for Windows.
- Release of the MVP artifacts with support for mixed-mode R and C/C++ stack sampling.
- Release of a version of **jointprof** using the new library to CRAN.
- Finally, release of e.g. blog posts, screencasts, or other public testimonials on the use of this project.

Future work

Our suggested future work would include

- Packaging of the command-line tool for various Linux distributions, first and foremost the Debian project (and downstream distributions, including Ubuntu). This would improve the ease with which this tool would be available in these environments.
- Support for macOS. Although it is unlikely that many users are running production workloads on macOS, it is a popular distribution for R developers.

- Support for Solaris and other less-popular platforms, such as FreeBSD.

Key risks

- It may turn out to be too difficult to support either R stack sampling or C/C++ stack sampling on Windows. This would be a major blow to the aims of the project, but the MVP of this proposal would still be deliverable.
- It may turn out to be too difficult or impossible to fully reconcile the R and C/C++ stacks when outputting mixed-mode samples. If this were to be the case, it should be possible to either fall back on R-only sampling or perform best-effort reconciliation.
- The current requirement to run the profiler with escalated priviledges may hamper its adoption or the possible ergonomics for end users.
- Lastly, the project could face significant delays if project member Aaron Jacobs is unable to work on implementing the deliverables. Should this come to pass, it may be possible to scope out the implementation work for assignment to another member of the community.