SCALING RAILS: A SYSTEM-WIDE APPROACH TO PERFORMANCE OPTIMIZATION

Gonçalo Santarém da Silva

Dissertation supervised by *Prof. Ademar Aguiar*
at *Escolinhas.pt*

1. Motivation and Objectives

The Web starts to play a critically important role in many people's lives, either from a professional or personal point of view. User experiences have become of great importance in recent times, with *Web 2.0* raising the expectations on better interactions. To help coping with this demand, many frameworks emerged. Ruby on Rails was one of these frameworks, which gained notorious popularity for being agile, robust and offering convenient methods and features which greatly improve the product's quality without the need for extended development times. However, it is often criticized for scalability issues.

There is the need for a solid set of general development conventions and guidelines oriented towards the scalability and performance of Rails projects. Developers seek optimal configurations for all the components involved so they can wring every processing cycle out of their applications, in order to increase their scalability and decrease their response times. There is urgency in looking at all components Rails depends on, determining which are best for which situation and tune them to suit Rails' needs—the system, envisioned as a whole.

Improving user experiences while profiling applications is also imperative. Profiling is a critical activity when optimizing applications but recent versions of Ruby and Rails break these tools and older versions only supported text-based outputs. Profiling should be functional, less verbose and more intuitive.

Finally, increasing the global awareness of the importance of building highly performant applications is also important. Developers should consider this aspect from the beginning and have access to all the information they need to plan, architecture and develop scalable systems.

2. Problem Statement

Some information regarding benchmarks and configurations of the various components involved in a Rails application exists but it is, however, very scattered. Most tests and configuration evaluations are not general, focusing on specific components, setups or purposes. It is crucial to create a solid set of general conventions and guidelines that cover all components, approaching benchmarking and tweaking from a unified perspective—optimizing a Ruby on Rails application.

On a related matter, the native profiling tools in Ruby on Rails have never worked properly when used with Ruby 1.9. While difficulting the act of profiling an application, this also prevents some users from switching to this version and leveraging from its increased performance. At the same time, the profiling output formats supported are all text based. Profiling is an essential aspect of increasing an application's performance, so it is very important to fix Ruby and Rails' profiling tools, provide a seamless integration between them and the most recent versions of Ruby and add support for alternate, more intuitive profiling output formats.

Twitter—and a few other renowned platforms—had many scalability issues which were not discrete, some becoming quite famous. However, the global awareness of the Ruby on Rails community regarding the importance of scalability is still remarkably low. Despite having the Rails 3 API available shortly after its development began back in early 2009 and Ruby 1.9 available since 2008, most developers have not added support for these versions to their plugins and gems. Famous Rails applications, like Redmine, also lack support for Ruby 1.9 and have not started upgrading to Rails 3. Knowing the performance benefits of using the latest versions of both components, it is important to increase the community's *momentum* by updating renowned plugins and applications, possibly igniting an update-focused philosophy.

This work's objectives are to create a general guideline with centralized information, improve the current profiling tools and increase the community's sensibility on this subject. For this to happen, it is crucial to address specific characteristics, configurations and issues of all components involved in a Rails-centered perspective.

3. Problem Approach and Results

To produce the aforementioned guidelines, greatly improve Rails-related profiling tools and increase this subjects' awareness inside the community all system components commonly found on Rails applications have been addressed from the aforementioned perspective. Each one of the following sections will expose the research and results achieved by working on each one of these components.

3.1. Operating Systems

Regarding development, a general benchmarking script oriented towards UNIX systems was created. On the benchmarking phase, this script was used on four Linux distributions—Debian, Ubuntu Server, CentOS and Gentoo—where CentOS underperformed and was discarded from future work. Then, a benchmark focused on web server performance was made using the three remaining Linux distributions. Gentoo yielded the best performance while maintaining a remarkable stability. It was then compared with FreeBSD, by using a renowned Ruby benchmarking suite. Having overall better performance, Gentoo was chosen to be the base for all future work. Finally, concerning tweaking, a few important kernel options were presented and their impact on performance was explained.

3.2. Ruby

Concerning benchmarking, the performance differences between YARV and its predecessor, MRI, were determined. As expected, YARV overall performance is noticeably better than MRI's. Regarding development, many activities were involved. First of all, *Escolinhas.pt* was ported to Ruby 1.9 to benefit from YARV's improvements. The porting effort was analyzed, concluding that it was significantly low. After that, YARV's GC flexibility was improved by creating five configuration options for adaptive performance. Initial tests were shown and by using dfifferent settings YARV's performance increased while consuming less memory overall. YARV's internal profiler was also improved. More data is being recorded, the *hash* format is now supported on result output and an HTML-based output format was incorporated, providing a graphical interface for easier analysis.

3.3. Rails Web Servers

As of development, a simple yet powerful memory usage monitoring script was created. Benchmarking, however, had many phases. First of all, proxy performance was evaluated, involving Apache, Nginx and Cherooke. The performance of all three was very similar, but Nginx used considerably less memory. Then, Passenger's performance on Apache and Nginx was compared. While having a similar performance across the mentioned web servers, Passenger was more scalable, more stable and used less memory under Nginx. After that, Thin and Unicorn were compared, where Thin performed slightly better while, once again, using less memory. Finally, a more complete benchmark involving Thin, Unicorn, Passenger, Nginx and different Ruby versions was made. All web servers yielded similar results, with Unicorn performing slightly better and Thin maintaining its low memory consumption. Switching from MRI to YARV, however, had a huge impact on performance, mainly noticed on the heaviest test. Finally, concerning tweaking, some web server options were presented and explained. Enabling threads in Thin was one of the options explored, though no performance benefits were detected.

3.4. Databases

A promising Ruby library for MySQL—*mysql2*—was improved, by changing its behavior concerning database field casting. Instead of converting all fields from MySQL types to Ruby objects upon fetching, it now lazy type casts them as needed.

3.5. Ruby on Rails

Regarding benchmarking, common performance pitfalls and their solutions were analyzed. Concerning development, Rails' profiling and benchmarking tools were revamped. After that, some renowned plugins and Redmine were ported to Rails 3. A plugin which adds support for Nginx's *X-Accel-Redirect* was also created. Finally, a project related to the creation of a performance-oriented continuous integration for Rails under the Ruby Summer of Code program was detailed. This section also presents some work that does not fit under the usual fields—benchmarking, development and tweaking—which is related with increasing the community's awareness of the importance of building highly performant Rails applications. This work is related with the creation of a community blog on this subject and the start of a series of performance-oriented articles for *Rails Magazine*.

4. Conclusions

General guidelines and conventions for building highly performant and scalable Ruby on Rails applications have been created. The conclusions drawn from the benchmarking phases of each component's analysis and their application to *Escolinhas.pt* allowed to fill in some the flaws found in the initial research, providing a solid knowledge base for the elaboration of the aforementioned guidelines and conventions.

The native profiling tools for Ruby and Rails applications have been revamped and brought up to date. Rails' profiling tools were refactored in order to enable support for YARV, the official interpreter for Ruby 1.9. On the other hand, improvements were also made on YARV concerning this subject, by recording new information and enhancing its information retrieval capabilities.

The global awareness of the importance in building highly performant and scalable Ruby on Rails applications also increased. Revamping the native profiling tools, adding configuration options to YARV's garbage collector, starting a public blog on this subject, writing a series of articles oriented towards performance for Rails Magazine and developing an official benchmarking continuous integration suite for Rails, all contribute to an increased awareness of this matter, from the core team of Rails to the web developers themselves. Given the performance advantages of the latest versions of Rails, porting Redmine to Rails 3 increases the visibility of these benefits since Redmine's users are able to experience it themselves. Finally, adding support for Ruby 1.9 and Rails 3 to some famous plugins also lowers the porting effort for some applications.