Exploiting the collective wisdom of web application executions

Brian Burg
University of Washington
burg@cs.washington.edu

1. Introduction

Web applications are quickly approaching the popularity and complexity of native applications. Software including video games, spreadsheets, word processors, and emulators have been implemented using HTML, DOM, and JavaScript. Recent developments such as WebOS, the Chrome App Store, and Mozilla Prism[1] have cemented the notion that web applications are *programs*, and their operating system is any web platform.

Web browsers make extensive use of ephemeral *runtime* optimizations, but rarely use profile-guided optimizations (PGO) based on multiple execution profiles. This is a consequence of frequently-updated web applications, the highly-dynamic nature of JavaScript programs, and a reluctance among vendors to increase the disk and memory footprint of browsers. These constraints have prevented browsers from using PGO-like techniques to improve performance.

These drawbacks can be mitigated by distributing the fruits and labors of profile-guided optimization on a much larger scale. Even after accounting for the quick upgrade cycle of web applications, still millions of people use popular web applications every day in the same execution environment. Collective knowledge about how these programs behave could be built by aggregating execution profiles created by many users. Distilling and distributing this knowledge changes the cost/benefit analysis of PGO use in browsers: the cost of creating profiles is spread thinly among many users, but the benefits of improved performance are reaped widely.

2. Collaborative Optimization

We envision *collaborative optimization* (alternatively, crowdsourced performance tuning) as a way to harness the collective wisdom of users running web applications under similar circumstances. Specifically, profiling data gathered at runtime and aggregated centrally can be exploited to optimize browser operations to the specific "program" being executed. Owing to web applications' highly-dynamic execution model, most collaborative optimizations use saved profile data to drive *dynamic* optimizations, as opposed the *static*, offline optimizations typically associated with the term "profile-guided optimization".

Each collaborative optimization has three parts: *collecting* interesting profile data, *aggregating* and analyzing profiles, and *exploiting* gathered information for optimization. Below, we sketch three hypothetical collaborative optimizations.

Resource request prioritization Browser vendors and users place importance on the load time of web pages; of particular interest is the time taken for the initial viewable area to be rendered and drawn in. This represents the delay before the user can begin reading the page. Web pages consist of dozens of discrete resources (such as images, stylesheets, video) in addition to structural markup and text. In most cases only a small proportion of these asynchronously-requested resources comprise the initial view of the webpage.

Unfortunately, browsers tend to request resources in the order of their appearance in the source, which in general has no relation to the visual proximity of that resource to the top of the page. One remedy is to aggregate the observation of many users (i.e., which resources actually appeared *visually* near the top of the page) and then use this collective wisdom to prioritize resource requests. This would improve the load-time performance of browsers for mobile devices with limited network and computational resources.

Speculating on dynamic behavior Dynamic language implementations conservatively check many guard conditions for each step of computation; reducing and prioritizing guards is key to improving average-case performance. Though JavaScript is highly dynamic, prior work has shown individual *uses* of dynamic features (i.e., call sites or allocation sites) to be consistent. For example, if an occurrence of `eval` only ever deserializes JSON, then the `eval` could be speculatively replaced with a safer, faster alternative. This technique generalizes to features such as exceptions, addition and deletion of fields, variable-arity invocations, and prototype mutation.

Biasing JIT compilation thresholds Tracing JIT compilers and method JIT compilers work by observing code paths and hot functions; empirically, it has been observed that these hot functions can be predictable across web application executions. The steps of tracing and compiling have a high cost which is only worth paying when it can be amortized over many subsequent optimized executions. Aggregated information of which code paths are reliably hot/cold, or do/do not usually make use of particular language features, can help JIT optimizations "break even" earlier and more reliably.

Techniques such as decision trees can help to segregate common but divergent program behaviors. Heavy use of a certain application feature will likely exercise different code paths than another feature, but the code is very likely to behave similarly for the same feature across multiple users. Other collaborative optimizations may also be segmented in order to provide a real benefit some of the time, as opposed to a negligible benefit all of the time.

3. Current and Future Work

Collaborative optimizations cannot exist in a vacuum; in order to deploy and evaluate these optimizations, there must be an extensible means for implementing each part of the optimization. Ongoing research projects at Mozilla aim to provide such a framework.

The optimizations themselves can be viewed as extensions and refinements of considerable prior work. Lightweight, extensible mechanisms for generating profile data are well-known, and were explored in previous work for browsers. A more challenging task is deciding *what* data is relevant and *where* within the browser it should be collected. Aggregation of execution profiles has been explored deeply in both static and dynamic contexts. Further research may be necessary to scale these techniques up to handle profiles at web-scale. Exploiting the collective wisdom of aggregated profiles is conceptually similar to other runtime optimizations.

Future work is concerned with the discovery, implementation, and evaluation of new collaborative optimizations. Each optimization admits several implementations which may differ in their approaches to user privacy and optimization aggressiveness.

[1] `http://prism.mozillalabs.com`