# Benchmarking Modern Web Browsers

Jordan Nielson Carey Williamson Martin Arlitt
Department of Computer Science
University of Calgary

***Abstract*— Many different Web browsers are available on the Internet, free of charge. A browser performs several tasks, such as rendering Web pages on the screen and executing client-side code often embedded in Web pages. Users typically choose a browser that gives them a satisfying browsing experience, which is partly determined by the speed of the browser. This paper presents benchmark performance test results for four popular browsers (Firefox, IE, Opera, and Safari) currently available on the Internet. The results indicate substantial differences among browsers across the range of tests considered, particularly in rendering speed and JavaScript string operation performance.**

## I. Introduction

The first graphical Web browser, Mosaic [4], was released in 1993, making the World Wide Web accessible to everyone, and helping to launch an information explosion that continues to this day. About a year later, Marc Andreesen founded Netscape, which released Navigator as its flagship product. In the following year, Microsoft joined the race by releasing its Web browser: Internet Explorer.

These events were the catalyst for what is commonly referred to as the "browser wars". Ever since, several companies have vied for the dominant share of the browser market. Even though the browsers themselves are not a great revenue stream, the browser is the "window to the Internet" for many users, and can be an influential factor in the choice of computing device, operating system, software, and services purchased by end users.

Given that most browsers are free, and (usually) functionally equivalent for displaying Web pages, how does a user select which Web browser to use? We argue that performance (i.e., responsiveness) is one of the factors influencing this decision.

Browser performance has garnered relatively little attention in the research literature to date, since the primary bottleneck has usually been elsewhere (e.g., server load, network congestion, TCP effects, round-trip latency). However, many of these performance problems have been effectively addressed (e.g., server clusters, proxy caching, persistent connections, parallel connections). More importantly, the advent of "Web 2.0" and the "services computing" paradigm have made the Web the preferred platform for numerous novel applications, and many of these rely heavily on client-side processing to facilitate interactivity for users and scalable deployments for service providers. Thus the ultimate success of "Web services" on the Internet will hinge on the user-perceived browsing experience.

The purpose of this paper is to present an apples-to-apples comparison of modern Web browsers, with respect to their browsing performance. We carry out this work experimentally, using four commonly-used Web browsers (Firefox, Internet Explorer, Opera, and Safari), and testing them with a small set of realistic micro-benchmarks.

The results show that no single Web browser is universally the best; in fact, there are noticable differences among the four browsers across the range of tests considered. For example, the rendering speeds for Web pages can differ by a factor of 2-3 across browsers, JavaScript string operator performance can differ by an order of magnitude, and some browsers show asymmetric performance for GET requests and POST requests in AJAX. These results are of value to users for browser selection, as well as to developers for browser performance improvements, and to Web service providers, for offering satisfying user experiences.

There are two main contributions in this paper. The first contribution is a direct performance comparison of modern Web browsers, with sufficient drill-down to the component level. This work represents a snapshot of current Web browser performance, providing an academic reference point to complement the ad hoc collection of anecdotal reports about browser performance available on the Web [3], [5], [6], [16], [23], [26]. A second (and perhaps more lasting) contribution is our experimental methodology, which can easily be applied to a broader set of browsers, or longitudinally to an evolving set of browsers over time.

The rest of the paper is organized as follows. Section II summarizes prior related work. Section III describes the structure of modern Web browsers. Section IV outlines our experimental methodology, and Section V presents experimental results. Finally, Section VI concludes the paper.

## II. Related Work

Performance-related research on Web user experience typically focuses on reducing the latency of page retrievals. Traditionally, retrieval of pages from remote Web servers was the primary performance bottleneck (either the remote server, the core network, or the user's access network). Two techniques are typically used to reduce retrieval latencies: *caching* and *prefetching*. Caches store copies of recently used pages, in case they are visited again in the near future. Much research has focussed on improving the management of such caches; for example, Jin and Bestavros proposed the GreedyDual* algorithm, and demonstrated its superiority to existing cache replacement policies [11]. Today, caches are deployed in browsers, proxies, and servers. Content Distribution Networks (e.g., Akamai) utilize caching at the "edges" of the Internet to increase the scalability of Web sites, as well as to reduce access latency for users.

Web prefetching algorithms (e.g., [9]) attempt to anticipate future Web page requests. Prefetching works in conjunction

with caching to try and hide retrieval latency from users. In the past, this topic garnered less attention than caching. However, with asynchronous requests in AJAX pages, the area may become more popular.

Research papers on benchmarking have focused on Web servers (e.g., [1]) and Web proxies (e.g., [18]). Results from Web browser studies tend to appear on developer sites, vendor product pages, and in white papers [3], [26]. Few of these articles are updated regularly, and none of them mention AJAX or AJAX-related performance (other than JavaScript). Research papers on Web browsers have focused more on new functionality (e.g., [15]) or new client devices (e.g., [7]) than on performance.

More comprehensive browser benchmarks will be necessary in the future, as browser performance becomes more important. A first step in that direction is characterization of changes in Web workloads. An example is the work of Schneider *et al.* [19], which characterizes Web 2.0 traffic.

## III. Modern Web Browsers

Modern Web browsers are composed of several parts. Each browser must have a *rendering engine* to create the layout and appearance of a Web page, a *scripting engine* to interpret and execute JavaScript (or similar) scripting code on a Web page, and a *user interface* that includes page navigation controls, as well as many other features (e.g., history, preferences, plugins) created by the browser designer.

When a typical user selects a preferred browser, they most likely base their decision on the interface, since it is the most visible distinguishing feature. However, the other aspects, which are more technical in nature, should not be ignored. Both the rendering engine and scripting engine could be graded on multiple aspects. In this paper, we focus solely on performance, leaving issues such as correctness and security aside.

Today's browser market consists of dozens of different choices. To limit the scope of the study, we consider four specific browsers. In alphabetical order, these are Firefox [13], Internet Explorer (IE) [12], Opera [17], and Safari [2]. These four browsers are currently the most popular ones on the Internet [20]. In April 2008, IE had the largest market segment (54.8%), followed by Firefox (39.1%), Safari (2.2%), and Opera (1.4%). While the exact usage numbers may vary depending on the source of the data, it is generally believed that these are currently the four most prevalent browsers.

Details on these browsers are provided in Table I. The table shows the rendering engine and JavaScript engine used in each of these four browsers. It also shows the current version of the browser used in our tests.

TABLE I
Summary of Web Browsers Tested

| Web Browser | Rendering Engine | JavaScript Engine | Tested Version |
|---|---|---|---|
| IE | Trident | JScript | 7.0.5730 |
| Firefox | Gecko | SpiderMonkey | 2.0.0.13 |
| Opera | Presto | linear_b | 9.26 |
| Safari | WebCore | JavaScriptCore | 3.0.4 |

### A. Rendering Engines

A rendering engine, also known as a layout engine [24], has the important task of displaying a Web page. Over time this task has become more complicated, with the ongoing evolution of the HTML standard, the continual addition of features, and the large-scale usage of Cascading Style Sheets (CSS).

The layout engine can be considered a separable component from the browser itself. For example, Mozilla Firefox's rendering engine, Gecko [14], and Internet Explorer's engine, Trident, are both used in a variety of other browsers and applications. Although there have been a number of layout engines developed, only four of them are usually used by current Web browsers. These include Gecko and Trident mentioned above, as well as Presto (used by Opera) and WebCore, which is the rendering component of WebKit [22], the engine currently used by Safari.

### B. Scripting Engines

Similar to a rendering engine, a scripting engine is also an important component of a Web browser. This engine's responsibility is to interpret JavaScript (or similar) code that is embedded in a Web page. Though a separable component, the scripting engine is often tied to its corresponding layout engine, since the scripts often influence the appearance of a Web page. Like layout engines, these scripting components have their own identities [25]. Firefox uses SpiderMonkey, Internet Explorer uses JScript, Safari uses JavaScriptCore (part of WebKit [22]), and Opera currently uses linear_b, but is switching to `futhark` in their upcoming version.

## IV. Benchmark Tests

Our experiments were conducted in a typical desktop PC environment, using the commonly-used Web browsers listed in Table I. We used the latest stable version available for each browser, along with the known bug fixes available at the time. All of the experiments were performed on an Intel Core2 CPU 6600 (dual core, 2.40 GHz), with 3 GB of RAM and running Windows XP Professional SP2.

To compare the performance of Web browsers, we use a set of benchmark tests. The tests are selected and designed in such a way as to exercise the typical tasks handled by a browser.

There are many factors that affect the performance of these browser operations. Therefore the goal was to design the tests to focus on the performance of the browser itself, while eliminating or isolating external factors (e.g., operating system, TCP implementation, network latency, server load). In addition, each test is meant to exercise a single specific element of the browser. However, a few exceptions had to be made. For example, to perform the timing and control of the rendering test, a snippet of JavaScript was required.

The purpose and specific details of each test are given next.

### A. Start-up

The simplest test considers browser start-up time. Since a Web browser is typically started only once per user session, the

start-up could be considered irrelevant. However, we include this test for completeness.

To automate the measurement, a script records the system time, and then launches the browser using a command-line argument to open a specific default page. This default page invokes a PHP script on the local machine that records the current system time (the time at which the page is accessed). The output from the script displays the elapsed time since the browser was launched.

### B. JavaScript

There are several client-side scripting languages supported by modern Web browsers, but JavaScript is the most commonly used and most universally supported. While JavaScript is not as powerful as some programming languages, it is still relatively complex. The browser is responsible for parsing and executing the JavaScript code, since it is a client-side language.

For JavaScript benchmarking, we used Apple's SunSpider JavaScript Benchmark [21]. This benchmark performs a thorough coverage of function calls, and is well recognized as a reliable measure of a browser's JavaScript performance.

The SunSpider test covers 9 aspects of JavaScript performance: 3D manipulation, access, bit operations, control flow, cryptography, date/time, mathematics, regular expressions, and string operations. The test was run using a local server to avoid significant network latency.

### C. Rendering

Rendering performance refers to the speed at which the rendering engine can layout and display all of the Web page objects within the browser window. For this experiment, we measure the time between requesting a page and the completion of the loading of the body of the page. The time is measured using JavaScript event handlers. The *onload* event of a document's body element is invoked automatically when a page has finished loading (rendering). By attaching a function call to the onload event, we can record the load time before proceeding to the next Web page in the benchmark.

The home pages of popular Web sites were chosen for this benchmark. We leveraged Dela Serna's comparison of traffic ratings services [8] to select the sites we used. To eliminate the impact of network latency to these sites, the pages were copied and stored on a local server. Care was taken to obtain the HTML and all of the linked files (e.g., images, CSS, and JavaScript), including background images defined in CSS files. A custom program was created to find and download these images automatically, and update the corresponding references, so that all benchmark tests could be executed locally.

One tricky issue was isolating the rendering portion of the loading time of the page. For example, most of these existing pages include a lot of JavaScript. To remove this impact, the benchmark preparation step removed all scripts and event triggers such as 'onload'. We also duplicated the content of the body element 10 times to increase the rendering load relative to the page overhead.

To perform the test a PHP script was created. This script determines the next page to be displayed based on the list provided to it. Inserted on the bottom of each page is a form with a placeholder for the load time of the current page. When the page is loaded, the placeholder is filled in with the current time and the form is submitted back to the PHP script. The script now sees the time for the current page, and delivers the next page in the queue. When all pages have been displayed sufficiently many times, the results are computed and displayed. The median load time for each page is the metric reported.

### D. AJAX

A recent emerging trend on the Web is the use of Asynchronous JavaScript and XML (AJAX). AJAX is not strictly a new technology, but rather a new method of using and combining existing Web technologies.

The way AJAX typically works is that the client side triggers the need for a refresh of some data on the Web page. While this could be done by refreshing the entire page, AJAX accomplishes this in a more efficient way by asynchronously retrieving data from the server, and updating the client's document accordingly. With AJAX, Web sites can deliver a smoother experience to the user.

The typical AJAX processing steps are as follows. A JavaScript function, often called by a timer or event, requests an XML object from the browser using HTTP. Using this object, the client sends a POST or GET request to the server (a PHP or other language script). The script on the server returns a response back to the browser, and the response is received by a previously specified JavaScript function. This function then updates the document based on the received response.

Our AJAX benchmark test measures each of these steps. The test serially sends many (alternating) GET and POST requests to the server, and follows each of these responses by updating the Document Object Model (DOM). To avoid caching effects, each request changes either the URL or the parameters for the GET/POST request. The test is performed on a local host, to limit the effect of network latencies.

## V. Experimental Results

### A. Start-up Test

The start-up test was run 6 times on each browser: 3 cold starts, and 3 warm starts. A cold start is when the browser is first loaded after booting the computer, while a warm start is when the browser is closed and then opened again soon. A warm start is often quicker, if part of the application is kept in memory.

The median start-up times are shown in Figure 1. The times were suitably low for all browsers (under 1 second), confirming the notion that start-up time is a non-issue. There was little difference between the cold start times and the warm start times for each browser. Nonetheless, the warm start times were slightly lower, likely due to caching done by the operating system.

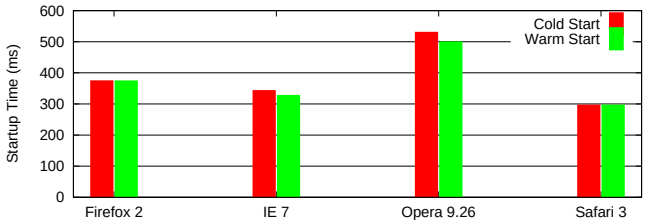



Fig. 1. Browser Start-up Times

### B. JavaScript Test

Each browser was tested using the SunSpider JavaScript test 3 times, with similar results each time. The results from these experiments are shown in Figure 2(a). (Note the logarithmic scale on the vertical axis in this plot.) The results of the test were not too surprising, however a few anomalies were observed across the 9 categories of JavaScript tests.

The fastest browser for this test was Safari, which averaged just under 8 seconds per run. Safari was the fastest in the categories of date/time, regular expressions, and string operations, with the latter category showing a substantial advantage. The next fastest overall was the Opera browser, which averaged just under 9 seconds per run. It was also the fastest in 5 of the 9 specific categories: 3D, access, bit operations, cryptography, and mathematics. However, its performance for string operations was much worse than that for Safari. Firefox had an average run time of about 12.5 seconds. It was the fastest in the control flow category, but performed quite poorly for 3D, bit operations, and date/time functions. Internet Explorer was the slowest overall, averaging 30 seconds per run. However, it did perform competitively in most of the categories. The primary difference was in string operations, where Internet Explorer took about 10 times longer than the others. Specifically, the JScript scripting engine in Internet Explorer performs poorly on the ‘base64’ and ‘validate-input’ tests in the string category [10].

### C. Rendering Test

The rendering test consisted of 5 iterations through the list of popular Web pages, as configured on the local machine. The browser’s cache was flushed before each test to control the effects of caching. While caching should not have much impact when all Web pages are local, some differences[1] could still arise. The median load time was used, to minimize the impact of any anomalous results.

Figure 2(b) shows the measurement results for 10 of the selected Web sites, in alphabetical order. The fastest browser in this test was Apple’s Safari, which was fastest not only overall, but also for loading each individual page tested. On average Safari took about 6 seconds to load all the pages. The second fastest browser in this experiment was Firefox with an average of about 19 seconds to load all of the pages. Close behind Firefox was Internet Explorer, with an average time of about 21 seconds. There were also some sites for which Internet Explorer was actually faster than Firefox. Opera took over 38 seconds to load the pages: almost twice as long as Firefox and Internet Explorer, and 6 times longer than Safari.

[1]In our experiments, the MSN Web page took 81 seconds to load initially in Internet Explorer, while it only took 1-2 seconds when it was cached. We do not yet have an explanation for this anomaly, which occurred only for this browser.

### D. AJAX Test

The AJAX experiments were performed with a local server just like most of the other tests, to eliminate the effect of network latency. The overall result simply measures the total time taken to complete the test, but we also provide a breakdown of the time taken for the individual components: GET requests, POST requests, and the DOM editing time.

The overall time is dominated by the GET and POST requests. Overall there were two “fast” browsers (Safari: 3.3 seconds; and Internet Explorer: 4.1 seconds) and two “slow” browsers (Firefox: 17.1 seconds; and Opera: 17.9 seconds).

The GET and POST categories show the same two-class behaviour. While there was no considerable difference between the time for GET requests and the time for POST requests for most browsers, Internet Explorer was approximately twice as fast on POST requests as on GET requests. Knowledge of this asymmetry could be useful to Web developers using AJAX, when choosing between the two different methods.

A finer-grained look at the results for DOM editing shows that Firefox is fastest, followed by Safari, Opera, and then Internet Explorer. While the other browsers took between 180 and 320 milliseconds, Firefox averaged about 3 milliseconds. While seemingly impressive, we are skeptical of this result. The abnormally low value may indicate that the work is delegated to a separate thread and done asynchronously. In other words, while the new thread is performing the actual work of editing the document, the original thread proceeds with recording the time, believing the work is done. Thus the DOM editing result should be interpreted carefully.

### E. Additional Tests

In addition to the four main Web browsers, we tested several other Trident-based or Gecko-based browsers. Their performance was comparable to their counterparts, as expected.

We conducted some throughput-oriented tests with bulk downloads and found no significant differences across browsers (e.g., 18 MB file download in about 160 seconds). Since all browsers are using the same native operating system and TCP implementation, this result is not unsurprising. However, we suspected that some browsers would be using download accelerators to gain a performance advantage; our experiments show no evidence of this.

Finally, we conducted some forward-looking tests. Firefox 3.01 and Opera 9.51 have recently been released, and Internet Explorer 8 is in the beta-testing stage (beta 1). We conducted a few tests with these browsers, to assess the robustness of our relative performance claims, as well as the claims made by the developers about significant performance improvements over the current versions.

The experimental results for these next generation browsers are shown in Figure 3. These graphs use the same vertical axes[2] as in Figure 2, to facilitate direct visual comparisons.

In most (but not all) cases, the new versions do show improved performance. All of the new browsers show substantial improvements in the JavaScript test, which was previously dominated by Safari and Opera. Firefox 3 now appears to be the fastest JavaScript browser overall, since it was not beaten in any category. Though still in early beta-testing stages, Internet Explorer 8 appears to have corrected some performance problems in its JavaScript implementation.

The rendering test also showed significant changes. Firefox improved its speed slightly, while Opera made a large improvement to catch up with the others. Internet Explorer 8 was actually significantly slower in this beta version, but it is probably too early to make any judgement about the rendering speed of the upcoming final release.

The AJAX test showed qualitativly similar results to the rendering test. Although Firefox 3 no longer reports a negligible document editing time (which was probably due to asynchronous behaviour), it still came out the fastest overall, slightly ahead of Safari. Opera also made some improvement, but is still the slowest among the browser versions tested.

## VI. Conclusion

This paper presented benchmark measurements evaluating the performance of modern Web browsers, primarily with respect to JavaScript, rendering, and AJAX performance. The experimental results show that there can be significant differences in performance between the different browsers. For JavaScript, the current versions of Opera and Safari are the fastest, with the next version of Firefox poised to take the lead. For AJAX applications, the current versions of Internet Explorer and Safari appear to be the fastest. However, once again the next version of Firefox achieves even better performance. For rendering speed, the current version of Safari is the fastest, followed by Firefox, Internet Explorer, and Opera, in that order. Assuming Internet Explorer 8's final release does not suffer degraded performance like the beta version, the new browsers would all be slightly quicker, but their relative ordering would remain the same.

We believe that there are numerous avenues for future work. The most obvious is evaluating browser performance for real examples of Web 2.0 applications, to see if there are quantifiable differences in user-perceived performance. As another example, the choice of operating system (and TCP stack) may affect the performance results. Understanding the performance implications of improved correctness (or security) is another open topic.

## Acknowledgements

The authors thank the anonymous HotWeb reviewers for their helpful comments on an earlier version of this paper. Financial support for this work was provided by Canada's Natural Sciences and Engineering Research Council (NSERC), as well as by the Informatics Circle of Research Excellence (iCORE) in the Province of Alberta.

[2]Note that one data point goes off the scale in Figure 3(b). The rendering time for the Facebook page in IE8 was 16.1 seconds.

## References

[1] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel, "Bottleneck Characterization of Dynamic Web Site Benchmarks". *Proceedings of 3rd IBM CAS Conference*, 2002.

[2] Apple Computers, "Apple – Safari". http://www.apple.com/safari/

[3] J. Atwood, "The Great Browser JavaScript Showdown". http://www.codinghorror.com/blog/archives/001023.html

[4] Boutell.com, "WWW FAQs: What was the first Web browser?". http://www.boutell.com/newfaq/history/fbrowser.html

[5] ConsumerSearch, "Web Browsers Reviews". http://www.consumersearch.com/www/internet/web-browser-reviews/

[6] CyberNet, "CyberNotes: Browser Performance Comparisons". http://cybernetnews.com/2008/03/26/cybernotes-browser-performance-comparisons

[7] O. de Bruijn, R. Spence and M. Chong, "RSVP Browser: Web Browsing on Small Screen Devices", *Personal and Ubiquitous Computing*, Vol. 6, No. 4, September 2002.

[8] A. Dela Serna, "Top 10 Most Popular Websites in the US". http://www.alleba.com/blog/2007/09/30/top-10-most-popular-Websites-in-the-us/

[9] D. Duchamp, "Prefetching Hyperlinks", *USENIX Symposium on Internet Technologies and Systems*, October 1999.

[10] Jaiprakash, "Performance Issues with String Concatenation in JScript". http://blogs.msdn.com/jscript/archive/2007/10/17/performance-issues-with-string-concatenation-in-jscript.aspx

[11] S. Jin and A. Bestavros, "GreedyDual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams", *5th Web Caching Workshop*, May 2000.

[12] Microsoft, "Internet Explorer Browser". http://www.microsoft.com/windows/products/winfamily/ie/

[13] Mozilla Foundation, "Firefox3/Firefox Requirements". http://wiki.mozilla.org/Firefox3/Firefox_Requirements

[14] Mozilla Foundation, "Mozilla Layout Engine". http://www.mozilla.org/newlayout/

[15] A. Nadamoto and K. Tanaka, "A Comparative Web Browser (CWB) for Browsing and Comparing Web Pages", *World-Wide Web Conference*, May 2003.

[16] Nontroppo, "Performance Tests for Opera 9.5". http://nontroppo.org/timer/kestrel_tests/

[17] Opera, "Opera Browser". http://www.opera.com/products/desktop/

[18] A. Rousskov and D. Wessels, "High-Performance Benchmarking with Web Polygraph", *Software: Practice and Experience*, Vol. 34, No. 2, January 2004.

[19] F. Schneider, S. Agarwal, T. Alpean and A. Feldmann, "The New Web: Characterizing AJAX Traffic", *Passive and Active Measurement Conference*, April 2008.

[20] W3Schools, "Browser Statistics". http://www.w3schools.com/browsers/browsers_stats.asp

[21] WebKit, "SunSpider JavaScript Benchmark". http://Webkit.org/perf/sunspider-0.9/sunspider.html

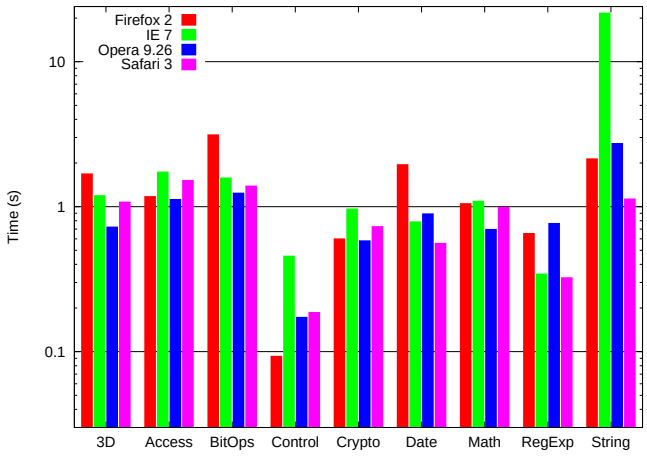
[22] WebKit, "Open Source WebKit", http://developer.apple.com/opensource/internet/webkit.html

[23] Web Performance Inc., "Safari 3 Windows Performance Analysis". http://www.webperformanceinc.com/library/reports/Safari%20Benchmarks/

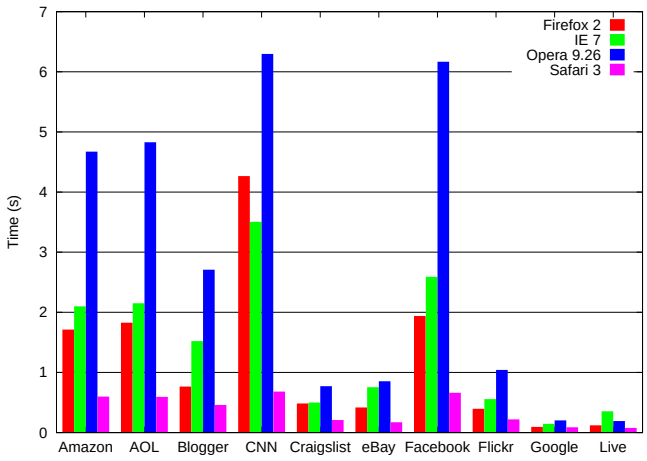
[24] Wikipedia, "Layout Engine", http://en.wikipedia.org/wiki/Layout_engine

[25] Wikipedia, "List of ECMAScript Engines", http://en.wikipedia.org/wiki/List_of_ECMAScript_engines

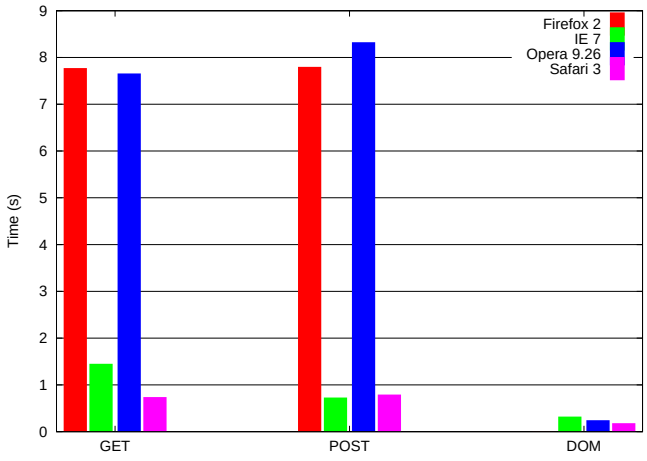
[26] M. Wilton-Jones, "Browser Speed Comparisons". http://www.howtocreate.co.uk/browserSpeed.html




(a) JavaScript

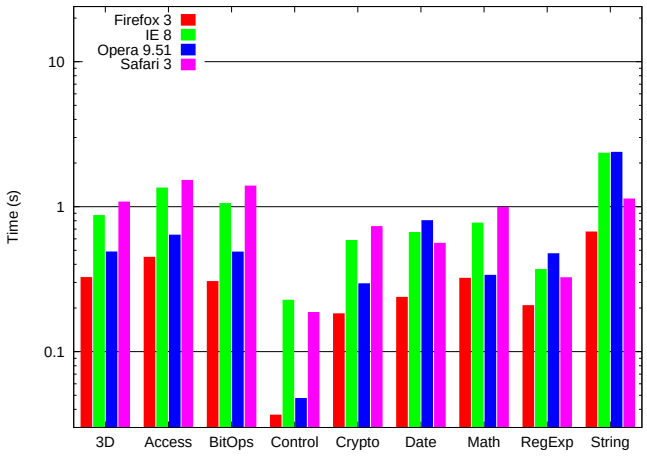



(b) Rendering

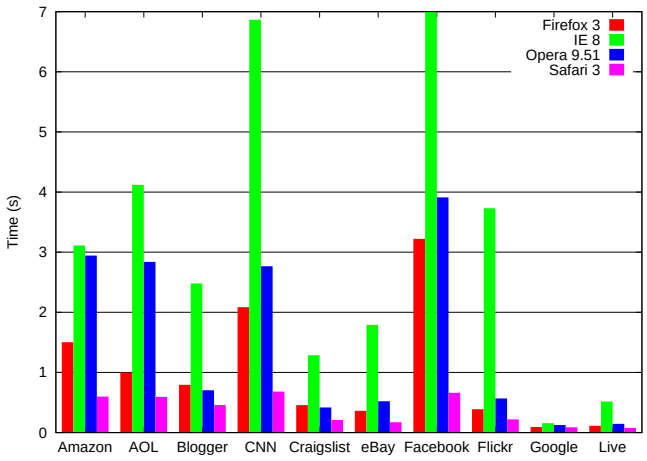



(c) AJAX

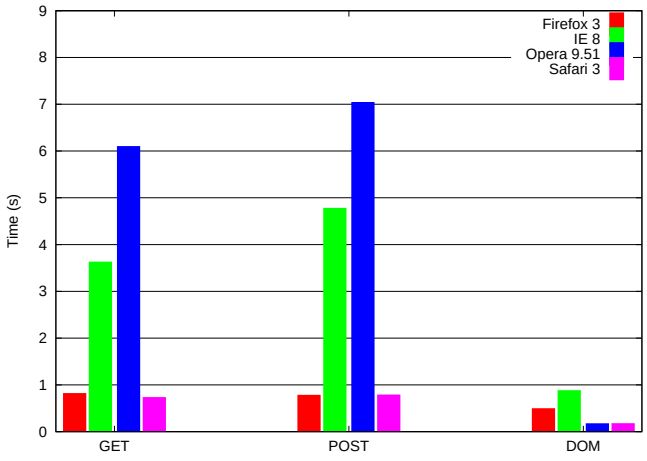
Fig. 2. Benchmark Performance Results for Current Web Browsers




(a) JavaScript




(b) Rendering




(c) AJAX

Fig. 3. Benchmark Performance Results for Imminent Web Browser Releases