Automatic Discovery of Performance and Energy Pitfalls in HTML and CSS

Extended Technical Report

Adrian Sampson, University of Washington
Călin Caşcaval, Qualcomm
Luis Ceze, University of Washington
Pablo Montesinos, Qualcomm
Dario Suarez Gracia, Universidad de Zaragoza

***Abstract*—Web browsers pose a unique challenge to performance and energy analysis tools: the complexity, variety, and volatility of implementations make it difficult to identify expensive aspects of Web content. WebChar is a tool for analyzing browsers holistically to discover properties of HTML and CSS that lead to poor performance and high energy consumption. WebChar analyzes a large collection of Web pages and builds a model for their performance based on static attributes of the content. It then mines this model for correlations between page properties and browser behavior. These correlations serve to suggest optimization opportunities for browser developers or design guidelines for content developers when targeting performance- and energy-constrained devices such as smartphones. An evaluation of WebChar on two platforms, a netbook and a smartphone, demonstrates that it can yield actionable yet unintuitive conclusions about the performance and energy consumption of Web browsers.**

I. Introduction

Web browsing is an increasingly important part of the end-user computing experience. The Web has begun to supplant the use of traditional desktop and mobile applications as the core technologies of HTML, CSS, and JavaScript gain similar capabilities to traditional toolkits but offer the advantages of a networked, cross-platform environment. Even in the performance- and energy-constrained mobile setting, the browser is important: a recent study suggested that smartphone users spend 33% of their time on their devices browsing the Web [1].

However, the performance and energy consumption of Web technologies limits their growth in the mobile space where these factors are most crucial. As a runtime system for mobile content and applications, the browser proves insufficient for many purposes: "native app" alternatives to mobile Web content still enjoy performance and battery-life advantages. To manage this gap, tools for understanding browsers' energy and performance characteristics are essential to creators of efficient Web content and to developers of mobile browsers.

Many recent studies and tools have focused on measuring and improving specific parts of the browser that are known to be bottlenecks, such as JavaScript execution [2]–[4] and the algorithms used for layout, font rendering, and CSS matching [5]. However, Web technologies are complex, browsers change rapidly, and real Web content often uses the technologies in unexpected ways [2], [3]. Moreover, while JavaScript is amenable to traditional tools like performance profilers, fewer tools exist to analyze the declarative languages HTML and CSS—even though they represent a large portion of browsers' execution time [5]. For this reason, developers currently depend on best practices and expert advice for information about browser bottlenecks [6]–[9]. A complete understanding of performance and energy on the Web requires empirical analysis of specific HTML and CSS implementations and constant adaptation to evolving standards and browsers, all in the context of real-world usage. Web optimization tools should give broad, up-to-date answers to these central questions: What makes some Web pages slower than others? Why do some sites seem to guzzle battery life while others sip it?

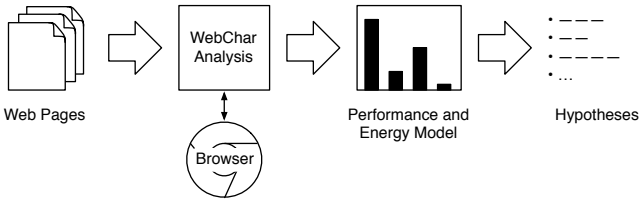

Fig. 1. WebChar analyzes a browser's performance and energy consumption when rendering a large set of Web pages to produce a model that describes the browser's behavior. It then mines this model to generate hypotheses concerning the browser's performance and energy pitfalls.

This paper describes WebChar (for *Web characterization*), a model-mining system for holistically analyzing browsers in the context of real-world HTML and CSS content. WebChar automatically identifies factors of popular Web pages that negatively impact performance and energy consumption. As Figure 1 depicts, WebChar analyzes a large body of HTML and CSS content to build a model that relates static page features to browser performance and energy consumption. WebChar then mines this model to discover detailed, non-intuitive potential browser performance and energy problems reflective of common usage. Web content developers can use WebChar results as recommendations to avoid certain content-authoring pitfalls; to browser developers, WebChar is a tool for discovering new high-level optimization opportunities.

In contrast to lower-level performance tuning tools like profilers, WebChar generates new high-level *hypotheses* about bottlenecks in browser performance and energy. These hypotheses reflect common trends among a diverse set of popular Web pages rather than for a single test input. Analysts can

then test these hypotheses using traditional benchmarking and profiling techniques to compose new recommendations to browser and content developers. WebChar addresses the problem of automatically generating best practices for both Web designers and browser developers, a process that has previously relied on the expert experience and guesswork.

This paper presents the general design of WebChar and details its specific implementation. We use it to analyze the performance and energy consumption of *in vitro* unmodified browsers running on two mobile systems, a netbook and a mobile phone. We distill a number of hypotheses from WebChar's output, some of which are familiar and some of which are nonintuitive. We use a series of microbenchmarks to test these hypotheses and demonstrate that WebChar's output yields new insight into browser performance. We detail a list of recommendations to Web browser and content developers that constitute new opportunities for browser optimization and best practices for efficient Web development.

II. Approach

Figure 2 summarizes WebChar's architecture. The WebChar system consists of two main components. First, a data collection module takes snapshots of a large set of popular Web sites, extracts data from the sites' code, and measures the page load time and energy for each site on a target browser. Next, an analysis step summarizes the page data into a set of numerical features and then builds a model that predicts browser performance (or energy) based on these features. Finally, WebChar mines this model to produce a ranked list of likely expensive features. The output of the workflow is a set of hypotheses reflecting potential best practices for Web designers and browser developers.

A. Data Collection

The data collection tool has three responsibilities: it downloads raw data from popular Web sites for analysis; it produces a simplified representation of each site's HTML and CSS content; and it measures the performance and power consumption of Web browsers while loading each page.

1) Snapshotting: To build a model, WebChar relies on a large body of real-world Web content. To obtain this input data, it would not suffice to download individual HTML documents from popular Web sites; instead, we need full *snapshots* of Web pages along with all linked linked content including embedded images and scripts. A page snapshot must include all information necessary to accurately replicate the experience of loading the page.

To capture a snapshot, we record an entire page-load session from the perspective of the network so that the process can be "replayed" later without involving remote hosts. Specifically, we instrument a full-featured, WebKit-based browser to load each page while recording every HTTP request and corresponding response. The responses are stored in an indexed database for efficient replay. This network-recording approach ensures that we can fully reproduce each page load process without accessing the network in order to study the browser in isolation.

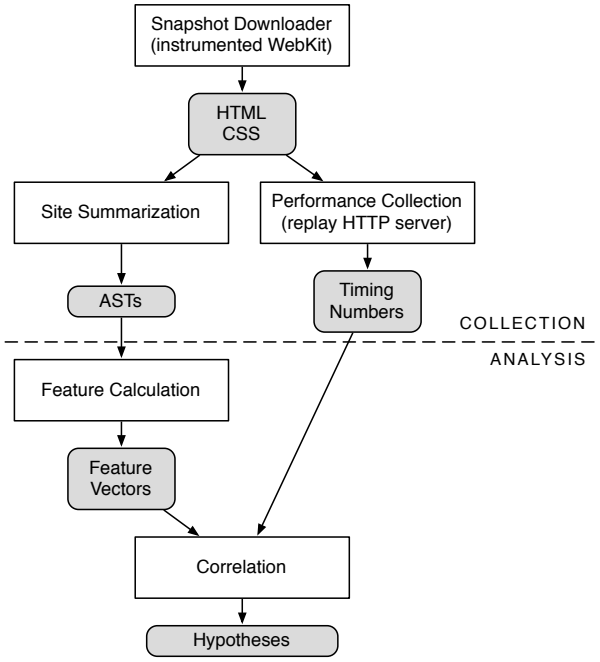

Fig. 2. Overview of WebChar's architecture.

2) Site Summarization: Given the raw network data for a page load, the summarization stage extracts relevant structures for each page that will be examined for performance and energy pitfalls. Specifically, WebChar examines each HTTP response to find those that contain HTML or CSS content. Then, using existing standards-based parsers for the two languages, the tool produces abstract syntax trees (ASTs) for later analysis. While raw HTTP responses are necessary for realistic performance measurement, parsed HTML and CSS ASTs facilitate meaningful analysis of each page's use of browser features.

3) Performance Collection: Finally, the system collects performance or energy metrics for a particular browser, OS, and hardware setup. The goal is to measure the amount of time or energy that the browser takes to display a page. In order to measure the behavior of the browser in isolation, we must eliminate the time and energy spent on network communication. Additionally, the measurement technique must work with any unmodified browser. This way, we can treat browsers as black boxes and observe them as they behave "in the wild" even when their source code is unavailable.

The performance measurement component consists of an HTTP server capable of replaying the snapshots collected earlier. The server receives a request, searches the snapshot database for a matching request observed during the record stage, and sends the corresponding HTTP response to the client. From the browser's perspective, the replay server is indistinguishable from a "real" remote server; the HTTP responses are identical to those of the original host.

To avoid network access, the server runs on the same machine as the browser under evaluation. The host system's DNS client configuration is modified to resolve all names to the

local host on the machine's loopback virtual network interface. In this way, we eliminate most of the effects of network latency from our performance measurement. Even without network access, some random error in the measurements is unavoidable; however, Section III-C quantifies the variance in measured page load times and shows that it is minimal when using this approach. That section also measures the overhead of the local HTTP server and finds it to be small.

To obtain accurate page load times without modifying the browser, the server must be notified when the load completes. To accomplish this, we modify each HTML snapshot to include callbacks to the server that fire when the page load completes (i.e., using the onload event handler). The server measures the page load time as the interval between receipt of the browser's first request and receipt of the callback. During the same interval, voltage and current readings are taken to calculate the page load energy (see Section III-B).

B. Analysis

Using the collected page data and performance measurements, WebChar's analysis component generates hypotheses about the performance and energy behavior of the Web browser under examination. First, WebChar summarizes the page data into a set of numerical *features*. It then correlates these feature values with the power and performance measurements to produce recommendations.

1) Feature Vector Calculation: Features are Web page metrics that could potentially correlate with performance or energy usage. In other words, each feature is a candidate for identification as an "expensive" aspect of Web content. WebChar's feature calculation stage produces a large number of *static* features from each page's HTML and CSS abstract syntax trees. We focus on static features in this work because they are entirely independent of the browser implementation; dynamic features, such as reflow events during rendering, are implementation-specific and an avenue for future work. Furthermore, WebChar recommendations that reflect static features are most useful during content development because they describe aspects of the code that content developers write. Recommendations based on white-box aspects of the browser's internal behavior are less likely to be helpful during content development.

The feature computation stage generates a large number of features indiscriminately; the correlation stage will pick out a small, human-analyzable subset that seem to cause performance or energy problems. Namely, the feature set consists of the relative frequencies of various AST elements:

- HTML tag types.
- CSS properties.
- Broad categories of CSS properties (borders, backgrounds, effects, etc.).
- Basic CSS selector types.
- CSS selector relationships (composition types).

Each of these counts is normalized to avoid giving undue weight to larger pages. For example, one feature is the fraction of HTML tags in the document that are <A> tags; another is the fraction of CSS property declarations that fall under the "border" category. In all, the present implementation calculates 253 features per page. While future implementations could consider more types of features, we find that this set of feature types leads to useful conclusions.

After this summarization, each page is represented as a vector of real numbers between 0.0 and 1.0. Each coordinate corresponds to a single feature.

2) Correlation: The correlation step is responsible for learning the relationship between each page's feature vector and its performance or energy consumption. Each page's feature vector and performance/energy data together constitute an example that WebChar uses to train its model. Any function estimation technique may be used to produce this model; WebChar uses support vector regression (SVR). SVR uses support vector machines to efficiently learn a high-dimensional hyperplane that reflects the relationship between inputs and outputs in the training data. For WebChar, the training data consists of the feature vectors (inputs) and the page load time or energy consumption (output). The SVR algorithm produces a function that interpolates and generalizes from these samples to predict the page load time or energy given a feature vector. This trained function is a model of Web browser performance: it captures the relationship between page features and their runtime cost.

Once the model is trained, we apply an optimization heuristic to find a maximum of the trained function—a feature vector that leads to maximally poor performance or high energy consumption. WebChar uses simulated annealing, a probabilistically robust global optimizer, as the optimization heuristic. The resulting optimized feature vector represents a set of feature values that, according to the trained model, result in bad browser behavior. In other words, the model predicts that a Web page exhibiting these features would perform worse than any real page in the training set.

The tool outputs the highest-weighted features in this pathological feature vector as hypothetical "expensive" features—candidates for optimization (by browser developers) or avoidance (by content developers). This ranked list constitutes WebChar's final output.

III. Evaluation

This section presents experiments we performed with WebChar to validate its utility in discovering new power and performance pitfalls in some sample Web browsers. The end-to-end system described in Section II, including data collection, analysis, correlation, and reporting, has been prototyped. The implementation used in this evaluation is available online at: http://sampa.cs.washington.edu/sampa/WebChar

The purpose of the evaluation is to demonstrate that WebChar can produce new, testable hypotheses that reflect power and performance properties that were previously not widely known; furthermore, we wish to show that many of these hypotheses can be verified experimentally. We begin by running the WebChar prototype with two browsing platforms to produce correlation results. We then distill these results into

hypotheses, some of which reflect known browser characteristics and many of which are new. We test each of the new hypotheses by measuring the resource consumption of a series of microbenchmarks. Using the hypotheses that are supported by experimental evidence, we make a set of recommendations to Web browser and content developers.

A. Tool Implementation

The snapshot collection tool described in Section II-A1 is based on the mirror tool developed as part of the QtWebKit open-source project.[1] The tool loads pages in a real WebKit-based browser and records the resulting HTTP traffic in an indexed database for later replay.

To measure page load times and control power consumption readings, a browser-based callback is used to inform the server when a page finishes rendering (see Section II-A3). To implement this callback, all HTML responses are augmented with code that adds a JavaScript handler for the <BODY> element's onload event. This handler redirects the browser to a known "dummy" URL, which causes a request to the replay server; the server stops its timer when this request is received. A more natural design might load pages in an <IFRAME> element and measure the load time in JavaScript, but many popular sites detect whether they are being loaded in an <IFRAME> and block access as a security measure. Relatedly, emerging Web standards describe an API for measuring browser performance directly [10], but implementations are not yet widely available.

In the correlation step, support vector regression (SVR) is used as the function approximation technique. The implementation comes from the PyML package.[2] Similarly, the simulated annealing implementation comes from the SciPy package.[3]

B. Methodology

We analyze data from a collection of pages from the 200 most popular sites as ranked by Alexa [11]. The data used in this evaluation comes from snapshots of the Web pages taken on June 13, 2011. We used an instrumented Web browser to record a snapshot of the pages and all their associated resources. For the present evaluation, each page in the dataset was loaded 10 times; we then took the mean load time and energy over these replications.

Figure 3 depicts the evaluation setup for the mobile phone. To avoid the overhead of off-device data transfer, we use a local HTTP server to "replay" the recorded content snapshot. Browser requests are redirected locally and avoid measuring network data transfer time. (While running a local HTTP server does introduce some overhead, we quantify this cost in the next section, III-C, and find it to be small.) The device communicates with a monitor system over USB to record timing events.

[1]QtWebKit mirror: http://trac.webkit.org/wiki/QtWebKitMirrorGuide
[2]PyML: http://pyml.sourceforge.net/
[3]SciPy: http://scipy.org/

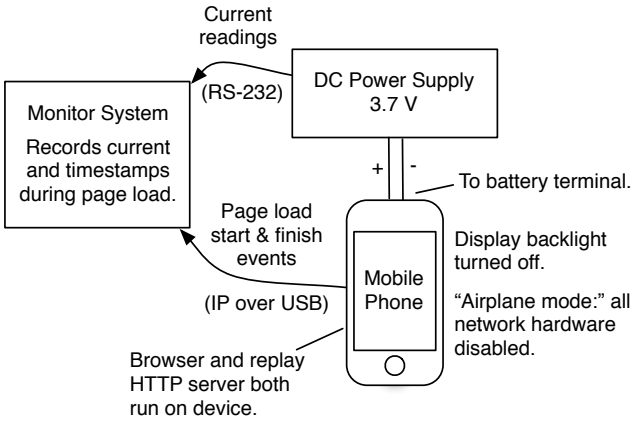

Fig. 3. Schematic of the mobile phone measurement setup for WebChar's evaluation. When measuring the desktop browser, the browser and server run on the monitor system and no power supply is used.

The monitor system also collects the mobile phone's energy consumption during page loads on the mobile phone. The device's battery was removed and replaced with leads to a DC power supply unit.[4] The phone's wireless communication hardware and display backlight were disabled to avoid measuring their power draw. While a page was loading, the instantaneous power consumption was measured repeatedly at approximately 26 Hz; this time series was integrated to compute a total *energy* for the page load. We also consider the average *power*, which is the total energy divided by the page load time.

Test Hardware: We measured two systems: one Android mobile phone and one Atom-based netbook. The mobile phone was a Motorola Droid handset running Android 2.3.2. The Droid had a Texas Instruments OMAP 3430 system-on-a-chip including a 550 MHz ARM Cortex A8 CPU and 256 MB of memory. We evaluated the Web browser included with the Android operating system, which is based on the WebKit browser library. The netbook was an Asus Aspire One D250 running Linux 2.6.39 with a 1.6 GHz Intel Atom N270 CPU (SMT enabled) and 1 GB of memory. On that platform, we evaluated Chromium 12.0.742.100, which is also based on WebKit. We measured the page load performance of both systems and the page load energy on the mobile phone, where whole-system power is most relevant.

C. Characterization

This section briefly details some findings from the performance and energy data collected for page loads. Table I summarizes some overall statistics.

Differences Between Machines: Even when compared to a low-power Atom-based system, the mobile phone used in our evaluation is at a significant performance disadvantage. The mean page load time for the phone was 2.7 seconds, about 3.5 times that of the netbook. These nontrivial page load times, which do not include network latency, suggest that browser

[4]BK Precision 1696: http://bkprecision.com/

	Minimum	Maximum	Mean
Netbook Time	30 ± 1 ms item.rakuten.co.jp	4690 ± 80 ms mail.ru	780 ± 30 ms
Phone Time	450 ± 30 ms es.youtube.com	26800 ± 300 ms auctions.yahoo.co.jp	2700 ± 100 ms
Phone Energy	0.49 ± 0.01 J viewmorepics.myspace.com	16.3 ± 0.6 J auctions.yahoo.co.jp	2.3 ± 0.1 J
Phone Power	0.51 ± 0.02 W twitter.com	1.39 ± 0.03 W es.youtube.com	1.07 ± 0.01 W

TABLE I. Overall statistics characterizing the performance and power of the page loads measured. Each page load was replicated 10 times; we report the mean and standard error over the trials. *Power* refers to the average power during the page load interval (energy over time).

performance still represents a substantial portion of browsing latency in mobile devices.

Power Variation: Before power measurements were taken, it was not clear whether the mobile phone's power consumption would be significantly affected by Web page content. If this were the case, measuring and analyzing energy for WebChar's correlation would be unnecessary—energy would be perfectly correlated with time ($E = P \times T$). However, we find that different pages can exhibit vastly different power consumption. The average power during a page load varied from 0.51 W to 1.39 W, more than a factor of two. This variation is not a result of random or unpredictable fluctuation; the standard error over 10 replicated page loads is typically within a few hundredths of a watt. As a result, we conclude that page load energy does not correlate perfectly with time; different factors influence the two resources differently. Section III-D explores the causes of this discrepancy.

Display and Radio Power: The energy measurements presented here exclude power spent on the mobile phone's display and network communication in order to focus on power consumed by the browser code itself. (As described in Section III-B, the radio and display backlight were disabled during measurement.) However, it is important to examine the importance of the browser in the context of these other power draws.

We measured the backlight's power consumption by varying its brightness—from completely disabled to maximum power—while measuring the full-system power. At full brightness, the LCD backlight on the phone draws about 0.13 W; the power draw scales approximately linearly with the brightness setting, so the backlight consumes about 0.06 W at medium brightness.

Similarly, we measured the impact of the device's wireless communication hardware by comparing the idle-state power (with all radios disabled) to the power consumed while transferring remote Web pages over Wi-Fi. Radio communication yielded an approximately 0.09 W difference in power consumption. (Longer-range wireless technologies, such as 3G data, are likely to be somewhat higher-power.)

The backlight and radio communication hardware each consume approximately 0.1 W. While these two components do make up a substantial portion of the whole-system Web browsing power, the CPU power consumption during browser execution can be significantly larger. As depicted in Table I, browser execution causes a power draw difference up to about 0.8 W. Browser software is, we conclude, just as important an avenue for power optimization as are hardware components such as display and radio devices.

HTTP Server Overhead: Our measurement technique uses a local Web server to avoid including network transfer in energy and time measurements. This technique does not eliminate data-loading costs entirely: namely, pages still need to be read from the device's internal flash memory. These flash accesses cost power that we then necessarily conflate with browser execution power. (An ideal measurement technique would use hypothetical fine-grained on-SoC power sensors to eliminate this factor.) However, some simple experiments show that loading data via our local HTTP server costs only up to about 0.1 W. This suggests that, while data loading does cost energy, it does not overwhelm the energy of computation. Furthermore, we find that the size (in bytes) of each Web page and its associated resources is not significantly correlated with energy or power, indicating that raw data volume is unlikely to be a confounding factor.

D. Correlation Results

Figure 4 depicts the output of WebChar's correlation and ranking. The lists represent page features that correlate with poor performance or energy on each system. The rankings contain features from the set of 253 features we collected for each page in the dataset (see Section II-B1); including more features could reveal more correlations.

From these raw rankings, we can distill a handful of testable hypotheses regarding the discovered page features:

1) The prominence of the <TABLE>, <TR>, <TD>, and <TBODY> elements, all components of HTML tables, suggests that heavy use of tables can cause poor performance.
2) Heavy use of images (reflected by the <IMG> HTML tag) can cause poor performance.
3) CSS selectors that match on classes and IDs are slower and cost more energy than matching by tag type.
4) The "descendant" CSS selector construction performs poorly on both platforms.
5) CSS effect properties (including shadows, rounded corners, and opacity) can cause performance problems, possibly due to added complexity in rendering.
6) Floating elements are slow, especially on the mobile phone.

Netbook performance:

1) CSS descendant selectors
2) <IMG> HTML tag
3) CSS border properties
4) <TR>
5) <DIV>
6) CSS ID selectors
7) CSS backgrounds
8) CSS effects
9) <TBODY>
10) CSS class selectors
11) <TD>
12) CSS text-align property
13)
14) <TABLE>

Mobile phone performance:

1) CSS class selectors
2) <DIV> HTML tag
3) <IMG>
4) <SPAN>
5) CSS padding attributes
6) CSS positioning
7) CSS float property
8) CSS effects
9) <TR>
10) <TD>
11) <P>
12) CSS descendant selectors
13) CSS display property
14) CSS padding-left property

Mobile phone energy:

1) CSS background properties
2) <IMG> HTML tag
3) CSS border properties
4) CSS ID selectors
5) <SPAN>
6) <OPTION>
7) <P>
8) <EM>
9) CSS background-position-x
10) CSS border-right-color
11) CSS background-origin
12) CSS background-position-y
13) CSS background-attachment
14) <STRONG>

Fig. 4. Correlation results for the two systems measured in our evaluation. Each list is a ranking of the most "expensive" page features inferred by WebChar—those features that the slowest-loading or most energy-intensive pages have in common.

7) Element backgrounds cost energy to draw on the mobile phone but may be less significant for performance.
8) The <SPAN> element costs energy and performance on the mobile phone.

There are also a number of ranked features that are harder to explain. For example, the unstyled <P> and <DIV> tags are highly ranked across the three metrics. Recall that WebChar discovers correlations and not causations—while these seemingly benign tags may be correlated with poor performance or high energy consumption, this could be because their use is correlated with a third factor. In our example, the predominance of these simple, unstyled tags may be accompanied with heavy use of CSS rules to customize their appearance and layout: the tags themselves may not themselves cause performance problems, but they may be correlated with other patterns that do.

The next section considers these hypotheses and tests empirically whether some of the more nonintuitive ones correspond to real performance and energy pitfalls.

E. Validation

In this section, we validate WebChar's effectiveness by determining the utility of the hypotheses listed in the previous section. Because WebChar reports correlations rather than proofs of causation, it is expected that some rankings it makes will not correspond to real performance bottlenecks. (For example, it is likely that the <DIV> tag is itself not responsible for poor performance; instead, sites with complex layouts may tend to use more of these tags.) WebChar is only useful if many of its recommendations are actually useful as evidence of performance and energy pitfalls.

We discuss each of the hypotheses listed in the previous section in turn, identifying the correlation results that are evidence of real performance and energy problems.

Some of the hypotheses correspond to known performance problems. The use of inline images (hypothesis #2) has a clear performance cost. CSS descendant selectors (hypothesis #4) are also known to be more costly than other selector styles [6]. These previously-documented pitfalls confirm WebChar's hypotheses; we now focus on those hypotheses that suggest new, unintuitive results.

These hypotheses—for example, the use of tables (hypothesis #1) and the application of CSS effects (hypothesis #5)—are not pervasively discussed as performance problems. To examine these hypotheses, we use a series of *microbenchmarks* constructed to test them. In each case, we construct a synthetic Web page meant to exercise a certain browser feature repeatedly. A "control" page is also constructed that uses the same structure but does not exercise the functionality in question. We measure the page load time and energy for each synthetic page and compare them.

For each microbenchmark, we load each synthetic document $N = 20$ times and take the mean page load time and energy. We perform independent two-sample t-tests of the statistical significance of the difference between these sample means.

Tables (Hypothesis #1): One HTML document contains 10 <TABLE> elements, each containing 100 <TR> (table row) elements with 5 <TD> (cell) elements each. A corresponding document is produced that uses <DIV> elements to replace the tables and rows and <SPAN> elements to replace the cells—the total number of elements and their nesting structure are kept constant.

The netbook's mean page load time was 14% slower when rendering tables than for the document without tables. This difference is statistically significant ($P < 0.01$). We failed to find a significant difference for either performance or energy on the mobile phone.

CSS Selectors (#3): Three documents were used, each containing 10,000 HTML elements and an equal number of CSS style rules; each rule matches exactly one element. Each document uses a different CSS selector type for the rules: one selects by tag name, another by ID, and a third selects by class

name.

Even with the large number of style rules, no significant differences were found.

Opacity (#5): Both HTML documents contain 500 <DIV> elements with a single style rule applied to all such elements. In one, the opacity attribute is set to 50% (opacity: 0.5;). In the other, the text color is set to black as a control (color: black;).

The document with the opacity effect applied was 8% slower on the netbook than the control document; on the mobile phone, it was 28% slower and cost 25% more energy. All three of these differences are significant at the 0.01 level.

Float (#6): Again, both documents contain 500 <DIV>s. In one page, the elements are all given the float: left CSS property. The other page applies a null style rule as above.

On the netbook, the page with floating elements took 5% more time to load. On the mobile phone, the page with floating elements took 16% more time and 17% more energy. Again, all three differences are significant ($P < 0.01$).

Background (#7): Elements in one page are given a colored background; elements in the other are not.

We failed to find a significant difference in page load time on the netbook or the phone. However, the energy consumption on the mobile phone yielded a significant difference ($P < 0.05$): the page with backgrounds cost 11% more energy than the page without.

SPAN Tag (#8): Three pages with 500 elements each were used. One page used <EM> elements, another used <STRONG> elements, and a third used style-neutral <SPAN> elements.

We found that <STRONG> elements are *cheaper* than unstyled <SPAN> tags on the mobile device: the former used about 14% less time and energy in our experiment ($P < 0.05$). No significant difference was observed on the netbook. We conclude that <SPAN> tags are particularly inefficient in the mobile browser implementation.

With the exception of the CSS selector benchmark, each experiment yielded at least one significant difference. This confirms that WebChar's generated hypotheses, which represent correlation but not causation, can lead to useful conclusions about energy and performance that *do* reflect causation.

The CSS selector hypothesis (#3) led to no useful empirical conclusions, suggesting that the correlation observed here was not a result of causation. One possible explanation for the observed correlation between class/ID selectors and poor performance is that these selectors are frequently used on pages with complex style structures that themselves incur performance costs. The presence of such "third factors" limits the viability of hypotheses produced by WebChar, underscoring the need for empirical validation.

F. Recommendations

The WebChar deployment detailed in this evaluation revealed some surprising conclusions. In this section, we detail these conclusions as they apply to browser and content development.

- Laying out tables can be expensive on the Chrome desktop browser. Unless tabular data must be displayed, content developers should avoid placing content into tables.
- The recently-introduced CSS opacity controls carry a significant performance impact, especially on the mobile browser we measured: translucent elements rendered 28% more slowly than opaque elements in our mobile phone tests. Web developers should avoid opacity effects; meanwhile, mobile browser developers should investigate using hardware-accelerated compositing for this feature.
- Across both platforms we measured, "floating" layout is expensive. Content developers should avoid it where possible; however, since modern Web page layouts frequently depend on these elements for their structure, browser implementors should optimize for this common pattern.
- Background fills cost significant energy on Android. Even when pages do not seem to load slowly on that platform, they may spend a large amount of energy drawing backgrounds. Mobile content developers should avoid using backgrounds where possible in order to conserve battery life.
- The Android browser exhibits a performance and energy penalty when using <SPAN> HTML elements. The developers should investigate the element's unnecessary inefficiency as its presence does not affect the page's appearance.

These recommendations represent examples of the possible advice that WebChar can help generate. Future deployments—using different browsers, new versions of the WebKit browsers considered here, new Web usage patterns, and broader feature sets—can continue to reveal new performance and energy problems.

Several of these findings are surprising and nonintuitive. The energy (but not performance) cost of element backgrounds, for instance, represents an unexpected discrepancy between performance and energy that can be exploited to conserve battery life on mobile devices. The cost of <SPAN> elements in the Android browser is also counterintuitive and is likely the result of a performance bug in the platform's implementation. These unexpected results are where WebChar is most useful: the technique can identify costly factors without relying on human intuition.

These recommendations do not suggest *reasons* for the performance and energy problems; traditional profiling tools should be used to carry out the browser optimization. A system like WebChar could also help narrow down the causes of performance pitfalls if its model were trained on *components* of the browser's runtime and energy consumption. For example, a specific correlation could be produced for the time spent in the browser's layout stage if this detailed performance data were collected. Such an extension would, however, require instrumentation of the browser; the present technique treats the browser as a black box and requires no modification.

From the perspective of browser implementors, the WebChar's feature data can itself be used to guide performance

tuning. One such optimization concerns the filtering of CSS traversal based on the number of distinct element IDs and classes in typical Web pages. CSS selector matching requires checking the entire path to the root of the DOM tree and is a performance bottleneck [12]. WebKit implements a Bloom filter to keep track of classes and IDs in order to avoid traversing the DOM tree when it can be proven that there cannot be any ancestor matching the selector of a given rule. According to WebChar's feature data, the median number of IDs and classes per page are 51 and 59 respectively and most pages define fewer than 200. Therefore one can optimize the size of the filter to capture the common case and trade off computation for memory savings when there is a page that exceeds this number.

IV. Future Work

WebChar is designed to be browser- and platform-agnostic, so future deployments of the tool using different browsers, different devices, and different performance metrics will yield more insights into Web performance.

The focus of this work is on externally visible, static page features that can be collected while treating the browser as a black box. By measuring internal, dynamic metrics of the browser's execution instead, we plan to use WebChar to generate detailed hypotheses that help browser developers diagnose pathologies. Dynamic features in this extension will include statistics collected during page layout and CSS matching in an instrumented browser.

Future work will also address making WebChar's analysis and results available to developers. Specifically, by coupling it with periodic collection of data from popular Web sites, WebChar can be used to discover and present trends in browser performance and energy consumption. A publicly-available Web service will display up-to-date and historical performance and energy analysis for a collection of browsers along with full performance and energy data for popular Web pages. The Web service will also expose WebChar's measurement infrastructure, allowing developers to measure the resource consumption of their own sites on the mobile phone testbed.

V. Related Work

Existing performance- and energy-related resources for content developers generally consist of best practices provided by experts [6]–[9]. While this advice relates predominantly to network optimization (advising HTTP compression, source "minification," and content delivery networks), some does address the behavior of the browser itself. For example, developers are advised against using CSS descendant selectors, leaving image dimensions unspecified, and placing JavaScript code in the document <HEAD>. These broad, experience-based recommendations fail to capture detailed, unintuitive, and implementation-specific performance and energy pitfalls that are a fact of life in the rapidly-evolving landscape of Web browsers. Tools like WebChar can make such best-practice advice more complete and allow it to grow and change along with the Web.

Some work has focused on new implementation techniques for Web technologies [13]–[17], including parallelization [4], [5], [12]. Even improved Web browsers, however, will likely exhibit hard-to-predict performance and energy pitfalls. WebChar is an automated system for exposing these pitfalls and guiding future research that repairs them.

Traditional performance and energy profiling tools are also relevant to the task of improving browser implementations. Profilers generally highlight "hotspots" in programs that are likely to be fruitful targets for optimization. While performance analysis at the level of individual functions is helpful for finding localized performance bugs, Web browsers are composed of many disparate components—a single performance pathology could arise from the interaction of the parsing, layout, and rendering components. For this reason, WebChar seeks to answer questions at a higher level: which aspects of the *input*—typical Web pages—can cause performance pathologies? WebChar's analysis is more directly relevant to content developers than is profiling data; to browser developers, it provides a high-level view of real-world performance that is not captured by a profiler.

Detailed analyses of JavaScript behavior [2], [3] are another source of advice for browser implementors. While this work does catalog characteristics of real-world Web content, it does not identify those characteristics that cause performance and energy problems in existing implementations. In this paper's terminology, these studies focus on *collecting features*—WebChar seeks to correlate these features with performance and energy usage to generate actionable recommendations.

Case studies [18] can also provide some insight into Web application workloads but do not scale to capture large and diverse portions of the Web. WebChar uses a data-mining approach to help automate the process of drawing conclusions from large collections of Web content.

WebChar's approach is similar to analyses for distributed systems that use data mining techniques to debug failures [19] and pathologies [20]. WebChar treats browsers as complex black-box systems and, like the prior work, analyzes externally observable metrics to infer internal problems.

VI. Conclusion

WebChar is a new technique for developing understanding of performance and energy in Web browsers, filling a crucial role in the improvement of the browsing experience on resource-constrained mobile devices. Using measurements from a wide variety of real-world Web content, WebChar correlates aspects of Web pages with their resource consumption in order to automatically generate hypotheses for developer analysis. WebChar seeks to help explain why some pages are faster or more energy-efficient than others.

By evaluating WebChar on two browsing platforms, we produced and validated a set of unintuitive findings that serve as recommendations to browser and content developers. WebChar eliminates the slow process of manually identifying bottlenecks from among the thousands of features that browsers implement. As the Web continues to grow as a

platform and as browser implementations continue to evolve, automated analyses like WebChar will be necessary to help identify new pitfalls when they appear.

REFERENCES

[1] D. Kellogg. (2011, Aug.) Mobile apps beat the mobile web among US Android smartphone users. http://blog.nielsen.com/nielsenwire/online_mobile/mobile-apps-beat-the-mobile-web-among-us-android-smartphone-users/.

[2] G. Richards, S. Lebresne, B. Burg, and J. Vitek, "An analysis of the dynamic behavior of JavaScript programs," in *PLDI*, 2010.

[3] P. Ratanaworabhan, B. Livshits, and B. G. Zorn, "JSMeter: comparing the behavior of JavaScript benchmarks with real web applications," in *WebApps*, 2010.

[4] E. Fortuna, O. Anderson, L. Ceze, and S. Eggers, "A limit study of JavaScript parallelism," in *IISWC*, 2010.

[5] L. A. Meyerovich and R. Bodik, "Fast and parallel webpage layout," in *WWW*, 2010.

[6] S. Souders, *High performance web sites: essential knowledge for frontend engineers: 14 steps to faster-loading web sites*. O'Reilly, 2007.

[7] ——, *Even faster web sites: performance best practices for web developers*. O'Reilly, 2009.

[8] Yahoo. Best practices for speeding up your web site. http://developer.yahoo.com/performance/rules.html.

[9] Google. Web performance best practices. http://code.google.com/speed/page-speed/docs/rules_intro.html.

[10] W3C. Navigation timing. http://w3.org/TR/navigation-timing/.

[11] Alexa top 500 global sites. http://www.alexa.com/topsites.

[12] C. Badea, M. R. Haghighat, A. Nicolau, and A. V. Veidenbaum, "Towards parallelizing the layout engine of Firefox," in *HotPar*, 2010.

[13] O. Buyukkokten, H. Garcia-Molina, A. Paepcke, and T. Winograd, "Power browser: efficient web browsing for PDAs," in *CHI*, 2000.

[14] J. Ha, S. C. M. R. Haghigha and, and K. S. McKinley, "A concurrent trace-based just-in-time compiler for single-threaded JavaScript," in *PESPMA*, 2009.

[15] M. Chang, E. Smith, R. Reitmaier, M. Bebenita, A. Gal, C. Wimmer, B. Eich, and M. Franz, "Tracing for Web 3.0: trace compilation for the next generation web applications," in *VEE*, 2009.

[16] J. Mickens, J. Elson, J. Howell, and J. Lorch, "Crom: faster web browsing using speculative execution," in *NSDI*, 2010.

[17] M. Dong and L. Zhong, "Chameleon: a color-adaptive web browser for mobile OLED displays," in *MobiSys*, 2011.

[18] S. Xu, B. Huang, J. Ding, and J. Dai, "Browser workload characterization for an Ajax-based commercial online service," in *IISWC*, 2009.

[19] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: problem determination in large, dynamic Internet services," in *DSN*, 2002.

[20] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance debugging for distributed systems of black boxes," in *SOSP*, 2003.