# Parallel Performance-Energy Predictive Modeling of Browsers: Case Study of Servo

Rohit Zambre*, Lars Bergstrom†, Laleh Aghababaie Beni*, Aparna Chandramowlishwaran*
*EECS, University of California, Irvine, CA
*ICS, University of California, Irvine, CA
†Mozilla Research, USA

***Abstract*—Mozilla Research is developing Servo, a parallel web browser engine, to exploit the benefits of parallelism and concurrency in the web rendering pipeline. Parallelization results in improved performance for *pinterest.com* but not for *google.com*. The overhead of creating, deleting, and coordinating parallel work can outweigh parallelism benefits. In this poster, we showcase results of our models, generated through supervised learning, that capture the relationship between key web page primitives and a browser's parallel performance. Such a model allows us to predict the degree of parallelism in a web page. Additionally, we consider energy usage trade-offs for different levels of speedups in our automated labeling algorithm. This is critical for improving the browser's performance and minimizing its energy usage. Experiments on a quad-core Intel Ivy Bridge (i7-3615QM) laptop show that we can improve performance and energy usage by up to 94.52% and 46.32% respectively on the 535 web pages considered in this study. Looking forward, we identify opportunities to apply this model to other stages of a browser's architecture and other performance- and energy-critical devices.**

## I. Introduction

For any particular browser, a content-heavy page (such as news sites) will take longer to load than a content-scarce one (such as search-engine sites).The workload of a web browser is dependent on the web page it is rendering. The page load times of web browsers have become a growing concern with the increase in complexity of web pages, especially when the user experience affects sales. This concern is even more acute on mobile devices. Under the same wireless network, mobile devices load pages 3× slower than desktops, often taking more than 10 seconds [5]. The web browser was not originally designed to handle the increased workload while still delivering a flicker-free experience to users.

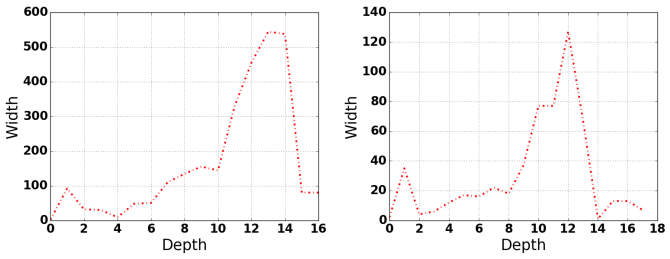
To address these concerns, Mozilla Research is developing Servo [4]: a new web browser engine designed to improve both memory safety, through its use of the Rust [3] language, and responsiveness, by increasing concurrency, with the goal of enabling parallelism in *all* parts of the web rendering pipeline.

Currently, Servo (Section III) naïvely parallelizes its tasks for all content. This may penalize very small workloads by increasing power usage or by delaying task-completion due to the overhead of coordinating parallel work. Thus, the challenge is to ensure fast, efficient page load times while preventing slowdowns caused by parallel overhead.

We tackle this challenge by modeling, using accurate labels and supervised learning, the relationship between web page characteristics and the parallel performance of a web rendering engine and its energy usage within the *complete* execution of a browser. Our modeling approach can easily extend to any parallel browser on any platform since our feature space is *blind* to the implementation of a web rendering engine. We evaluate our model for Servo on a quad-core Intel Ivy Bridge using off-the-shelf supervised learning methods on 535 web pages (from the Alexa Top 500 [2] and 2012 Fortune 1000 [1] lists). Even with a large class-imbalance in our data set, we demonstrate strong accuracies, approaching 88%. Concretely, we make three main contributions:

1. **Workload characterization** (Section II).
2. **Performance-energy labeling of web pages** (Section IV).
3. **Performance-energy predictive modeling** (Section V).

## II. Web Page Characterization




(a) `samsclub.com` (b) `westlake.com`

Fig. 1. Width Vs. Depth graphs of the DOM trees of different web pages (note that the scales of the axes are different).

The DOM tree is an object representation of a web page's HTML markup. The DOM tree is traversed extensively to compute the styles for each element on the page. Any amount of parallel speedup or slowdown would depend on the structure of the DOM tree. We intuitively choose the following set of seven characteristics to capture the properties of a web page and its DOM tree:

1. Total number of nodes in the DOM tree (**DOM-size**)
2. Total number of attributes in the HTML tags used to describe the web page (**attribute-count**)
3. Size of the web page's HTML in bytes (**web-page-size**)
4. Number of leaves in the tree (**number-of-leaves**)
5. Average number of nodes at each level (**avg-tree-width**)
6. Largest number of nodes at a level (**max-tree-width**)
7. Average work per level of the tree (**avg-work-per-level**)

Our intuition is that DOM trees with high values of these features are good candidates for parallelism. Figure 1 shows that `samsclub.com` is "good" for parallelism while

`westlake.com` is "bad" for parallelism. This is evident from our dataset as well. These characteristics serve as input features for our predictive model (Section V).

## III. Servo Overview

Currently, Servo parallelizes its *Styling* and *Layout* stages amongst its web rendering tasks and it does so blindly for all web pages (without considering any of their characteristics).

*Styling* is the most time-consuming web processing task. Servo uses parallel tree traversals, an approach similar to the one used by Meyerovich et al. [7]—each DOM-tree-node's style can be determined at the same time as any other node's. This is the first stage analyzed in our work.

The second stage that we analyze is *Layout* which determines the final geometric positions of all elements on the page. Servo executes consecutive, parallel top-down and bottom-up tree traversals for this stage; the height of a parent node is dependent on the cumulative height of its children, and the widths of the children is dependent on the width of the parent.

A work-stealing scheduler is used to schedule the threads that are spawned (once) to perform the parallel tree traversals in the *Styling* and *Layout* stages.

## IV. Performance and Energy Automated Labeling

We propose tunable, automated labeling algorithms that can be used with any web browser on any testing platform in order to classify web pages into different categories. We first collect performance and energy data, for each web page, using $N$ thread configurations. Then, for each page, we compute speedups, $p_t$ and greenups, $e_t$ [6] for each thread configuration, $t$. Using a cost model, we label each page using one of the $N$ labels. We consider three cost models (refer to poster) but use the labels generated by the *Performance and Energy Cost Model* to train our model using supervised learning methods (Section V). We define the following terms:

- $PET_t$ – *performance-energy tuple (PET)*, $\{ p_t, e_t \}$ defining the speedup and greenup achieved using $t$ threads.
- $P_jP_{j+1}$ – *PET bucket* comprising a certain number of PETs. $P_j$ and $P_{j+1}$ are speedup values. $PET_t \in P_jP_{j+1}$ if $P_j < p_t < P_{j+1}$. One can define an arbitrary number of such buckets to categorize the tuples.
- $E_j$ – energy usage increase limit (defined in terms of greenup) for $P_jP_{j+1}$. $E_j$ demarcates the tolerance of energy usage increase for all $PET_t \in P_jP_{j+1}$.

For our study with Servo on the quad-core Intel Ivy Bridge, we use three thread-configurations: 1, 2 and 4 threads. To account for noise, we set $p_{\min} = 1.1$. Using two PET buckets for the *Performance and Energy Cost Model*, the total number of web pages categorized into labels 1, 2, and 4 are 317 (59.25%), 50 (9.34%), and 168 (31.40%) respectively.

## V. Performance Modeling and Prediction

Using the labels generated for the *Performance and Energy Cost Model*, we construct predictive models for Servo using off-the-shelf supervised learning methods. The output response for each of the models, in our case study, is one of the three labels: 1, 2, or 4 representing 1, 2, and 4 threads respectively.

Our data set has a *large* class imbalance *i.e.* one label has many more observations than the other: 49 instances are labeled 2 while 299 and 187 are labeled 1 and 4 respectively. Additionally, since the predictor to observation ratio for our data set is quite small, we don't face the issue of over-fitting.

To validate and test our models, we use *cross-validation* using a 90-10% training-testing ratio which results in 10 model prediction accuracies. We consider the the mean and maximum of these 10 accuracies. When we are not able to use cross-validation, we use the *holdout-set* technique.

We train our models using Multinomial Logistic Regression (MNR), Ensemble Learning, and Neural Networks to capture non-linear relationships between web page characteristics and a rendering engine's parallel performance. Without tweaking the parameters of these machine learning techniques, we observe strong accuracies for all models with MNR achieving the highest accuracy of 87.27%. This emphasizes the effectiveness of our automated labeling algorithms. The MNR model, compared to the current implementation of Servo, achieves a maximum of 94.52% performance savings (on `indeed.com`) and a maximum of 46.32% energy savings (on `starbucks.com`).

## VI. Conclusion

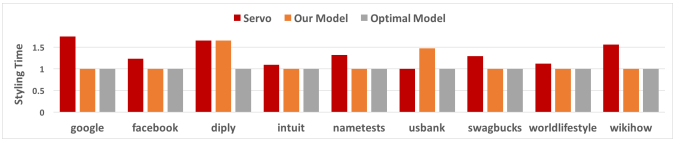



Fig. 2. Normalized styling times of Servo, Our Model and Optimal Model.

We confidently model the relationship between key web page features, using DOM tree features, and the parallel performance of the rendering engine using supervised learning methods. We propose accurate and tunable automated labeling algorithms that categorize web pages into a user-defined number of classes. More importantly, our algorithm accounts for trade-offs between speedups and energy usage increases for multi-core processors. We demonstrate strong predictive model accuracies, achieving 87.27% with 535 web pages within the complete execution of a browser. On a laptop platform, our MNR model achieves performance and energy savings up to 94.52% and 46.32% respectively. Figure 2 is a snapshot of styling times taken by *Servo* and *Our Model* against an *Optimal Model* (to which times are normalized) for web pages from our dataset. Overall, *Our Model* performs as well as the *Optimal Model* and better than *Servo* in 72% of the cases.

## References


[1] 2012 Fortune 1000. http://booleanstrings.com/wp-content/uploads/2014/01/fortune1000-2012.xls.

[2] Alexa top 500. http://www.alexa.com/topsites/countries/US.

[3] The Rust language. http://www.rust-lang.org/.

[4] The Servo web browser engine. https://github.com/servo/servo.

[5] M. Butkiewicz, D. Wang, Z. Wu, H. V. Madhyastha, and V. Sekar. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *12th USENIX NSDI 15*, pages 439–453, 2015.

[6] J. Choi, D. Bedard, R. Fowler, and R. Vuduc. A roofline model of energy. In *IPDPS, 2013 IEEE 27th Intl. Symp. on*, pages 661–672. IEEE, 2013.

[7] L. A. Meyerovich and R. Bodik. Fast and parallel webpage layout. In *Proc. of the 19th Intl. Conf. on WWW*, pages 711–720. ACM, 2010.