Energy Efficient Web Surfing for Smart Phones

Mulik Umesh, Bora Sanket, Chavan Ankita, Bande Jyoti

Abstract— Now-a-days we have observed that a lot of power is wasted during web browsing on mobile phones due to the special characteristics of wireless radio interface. Accordingly, we have identified these characteristics and have also taken into consideration the power consumption issues with the help of two techniques. Firstly, we analyze the energy consumption of the Android browser at popular web sites such as Facebook, Amazon, and many others. Secondly, to measure the energy needed to render individual web elements, such as cascade style sheets (CSS), JavaScript, HTML, images, and plug-in objects and also how to design web pages so as to minimize the energy needed to render the page. After that shows the results in the form graph and table format with related to those web elements. Our approach can reduce the power consumption and reduce loading time while surfing on the Android phone web browser.

Index Terms— Android, mobile computing, mobile web browser, offloading computations, power consumption, portable devices, wireless communication.

1 Introduction

Web surfing is one of the most important and commonly used service provided by smart phones. However, the current smart phone web browser wastes a lot of power when downloading web pages. [1]. But another issue in the smartphone Cellular networks required high energy consumption on the mobile devices due to the radio resource allocation operation performed at the operator end, which differs from other technologies such as Wi-Fi. Most applications are unaware of the transmission energy characteristics of cellular networks, where the energy consumption is not proportional to the amount of data sent: small data transfers can consume as much as a fully utilized link using UMTS 3G networks and 4G LTE network. There are various computations when loading a webpage such as HTML parsing, JavaScript code execution, image decoding, style formatting, page layout, etc. So, we measured the energy needed to render particular web sites as well as the energy needed to render individual web elements such as images, JavaScript, and Cascade Style Sheets (CSS) [1]. Our approach can reduce power consumption on smartphone and also increase network capacity. Moreover, our solution can reduce the page loading time while surfing on the web browser with also shows the results in the format of graph and table. In Section 2, authors would be discussing existing literature related that workflow of webpage processing in smartphone based web browser. Section 3 would comprise of implementation methodology and details of various algorithms to be used in the proposed approach. Section 4 contains the results obtained by following the approach. The paper would then end with a suitable conclusion in Section 5. Moreover, in section 6, authors discuss.

- *Mulik Umesh, Bora Sanket, Chavan Ankita and Bande Jyoti is currently pursuing degree in computer engineering in Pune Universty,Maharashtra, India, MB- +91 9960634734/9673340654 .*
- *E-mail: umesh.mulik.1111@mail.com, sanketbora1994@gmail.com*
- *Mr.Suhas Patil, Assistant professor Computer engineering department, in K.J College of engineering and management research, Pune, Maharashtra, India, MB- +91 9860647213. E-mail: patil.suhas16@gmail.com*

2 Background

In this section, we introduce the background knowledge related to power consumption of the 3G radio interface and browser processing steps [1] [3].

2.1 Power Consumption of the 3G Radio Interface

To efficiently utilize the limited radio resource of the backbone network, the 3G Radio Resource Control protocol defines the following three states for smart phones to control their radio interface as follows:

- IDEL State: This State Slightly consumes Power. Smart Phone can't send user data due to non-signaling connections with backbone network.
- DCH State: In this state, Smartphone can send user data because dedicated transmission channels to the Smartphone, it require more power.
- FACH State: Compare to IDEL and DCH State it require low speed to transmit user data and signaling data through common shared transmission channels [1].

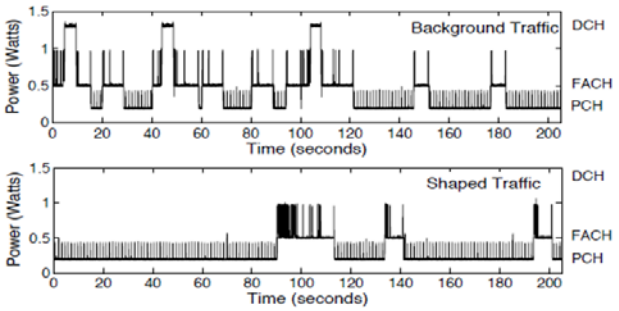

Fig. 1: Energy consumption examples of background traffic and energy-efficient shaped background traffic.

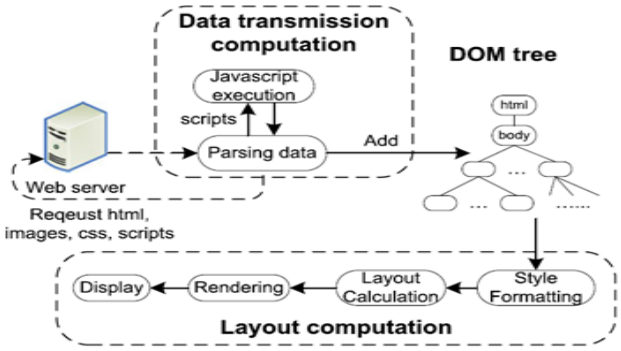

Fig. 2. The workflow of webpage processing in smartphone based web browser.

As shown in Fig. 2, in the latest smartphone web browser, there are two types of calculations combined with each incoming object. The first type is the calculations that achieve new data transmissions such as HTML and CSS file parsing and JavaScript code execution, which are commonly known as the data transmission computation. The second type is the computation that does not cause data transmission. This type of computation is used to lay out the webpage such as image decoding, style formatting, page layout calculation and page rendering, which is referred to as the layout computation. Images, JavaScript, and CSS are the power eager web components of a web page. We have to enhance web pages so as to curtail the power consumption of these elements.

1) Reducing JavaScript Power Consumption JavaScript is the most energy absorbing components of a web page. Websites using JavaScript requires a huge download and energy. This happens because of these webpages load large JavaScript files for delivering the web pages to users even though there is no need of all script for loading web page.

The Wikipedia webpage attached with two JavaScript as application.js and jquery.js. The application.js JavaScript is precise to the Wikipedia site and the jquery.js JavaScript is precise to the generic jquery JavaScript library. Each section of Wikipedia page such as introduction, history, Table of Contents, etc. can be crash and developed by the click of a button above each section of a page. The jquery.js JavaScript is mainly needed to dynamically describe the correct section depending on the id of the button clicked. But loading of this single JavaScript in to the memory requires 4 Joules of energy.

Reducing JavaScript on a mobile page to include only those functions required by the page highly decrease energy use. Generic JavaScript libraries simplify web development, but highly increases the energy used by the resulting pages [2].

2) Reducing CSS Power Consumption

Huge CSS files with unwanted CSS rules absorb more energy than minimum needed energy. Apple absorbs a large amount of energy to download and deliver CSS. It requires 12 Joules of energy for downloading and delivering purpose. Reason behind this is that Apple homepage uses 5 different CSS files containing distinct rules used in the page.

The logic to avoid energy consumption is to replace multiple CSS files by single file which consists of all rules required for the webpage. This idea conclude energy drop up to 5 Joules. This energy consumption can be saved by using single CSS file with only the required CSS rules. CSS file should be page precise and must include only those rules which are useful to elements in the page [2].

3) Image Formats: Comparison and Optimization Most commonly, web sites uses various types of image formats, as JPEG, GIF, and PNG. But the energy consumption to deliver an image to user totally depends on the setting of encoding format. We can easily compare the energy consumptions of the different types of image format. Let us consider 3 dominant image formats as PNG, JPEG and GIF. GIF format supports 8-bits per pixel and uses the Lempel-Ziv-Welch (LZW) lossless data compression method. PNG is uses a bitmapped image format that was invented to boost upon and replace GIF. PNG also supports for lossless data compression. JPEG is another prominent image formats that supporting lossy data compression. In mobile web browsing, mostly, GIFs are used for very small size images, and PNGs and JPEGs are used for larger images. JPEG is the more energy efficient format on the Android mobile phone for all image sizes [2], [4].

3 PROPOSED SYSTEM

3.1 Proposed Architecture

Following figure shows the system architecture of proposed system:

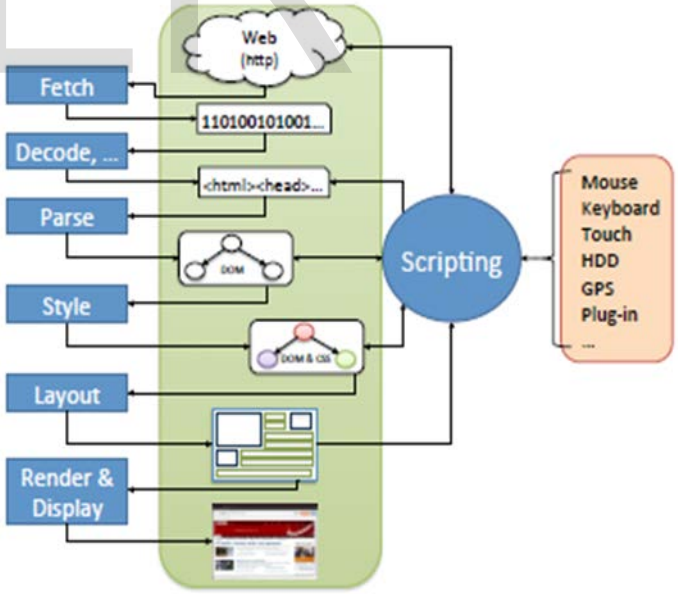

Fig. 3. Typical browser processing steps.

Figure 3 shows typical browser architecture and illustrates the steps required to render a web page. JavaScript interacts with the page during the page load, as well as after the page is loaded to provide interactivity. Our measurements, similar to show the network time being 30%-50% of the total execution time. Given this breakdown of computation, it is clear that in order to optimize the execution of the browser; one has to ad-

dress all components [1],[3].

3.2 Design goals

Our goal is to exploit concurrency at multiple levels: parallel algorithms for individual passes to speed up processing of each component, and overlapping of passes to speed up total execution time. In addition, we must respect the HTML and JavaScript semantics, even during concurrent execution. The main data structure that is used by all browser passes is the Document Object Model (DOM). The DOM is a tree representing all the HTML elements: their contents, relationships, styles, and positions. Web programmers use JavaScript to manipulate the DOM, producing interactive web pages and web apps. Most communication between browser passes and components happens through the DOM [3].

4 METHODOLOGY

4.1 Parallel DOM Styling

DOM styling is the means by which the CSS engine uses in memory rules to determine the style of the nodes in the DOM tree. For each node, the CSS engine must first find all the rules whose selectors match the node, or rule matching. Rule matching often returns many — and usually conflicting — rules per node. Using cascading, the CSS engine assigns weights to rules and chooses only the ones with the greatest weight. During style creation, the CSS engine creates the style data structure using the rules selected by the cascading algorithm and attaches it to the node. A key insight is that it is possible to concurrently style several DOM nodes as long as certain dependencies are enforced, and we developed a new parallel DOM styling algorithm that leverages it.Algorithm 1 shows that the CSS engine uses two types of tasks per node to style the DOM tree: matching tasks and styling tasks.

Styling tasks apply the cascading algorithm and create the final style data structure for each node. A styling task must satisfy two dependencies before it can execute. First, it can only execute after the matching task working on the same node has completed execution, since the cascading algorithm uses the rules selected by the matching task. Second, a styling task working on a node can only execute after the styling task working on the node's parent has completed execution. This is because some style properties of a node may be inherited from its parent. By using two types of tasks, our algorithm cans rule-match a node before its parent is styled. This basic version of the algorithm limits style sharing to parent child sharing. However, we subdivide style objects into sub styles containing related properties, and allow sharing at the sub style level, which increases the degree of available sharing. For example, if a child node uses the same font properties as its parent, then they can share the font sub style. To add support for sibling style sharing, the matching task must speculate whether a child node may be able to share its style with a previous child node before spawning tasks for it. If the answer is yes, it does not spawn new tasks for the second child [1], [3].

4.2 ALGORITHMS USED

Algorithm 1 Parallel DOM Styling Algorithm

```
1: function StyleNode(node, ruleset)
2:     finalRuleset ← Cascade(ruleset, node)
3:     node.style ← BuildStyle(finalRuleset, node)
4: end function
5:
6: function MatchNode(node, ruleset, styling)
7:     child ← node.FirstChild()
8:     while child ≠ NULL do
9:         r ← NewRuleset()
10:        ts ← NewTask(StyleNode(child, r))
11:        tm ← NewTask(MatchNode(child, r, ts))
12:        tm.SetSuccessor(ts)
13:        styling.SetSuccessor(ts)
14:        tm.Spawn()
15:        ts.Spawn()
16:        child ← child.NextSibling()
17:    end while
18:    Match(node, ruleset)
19: end function
20:
21: function StyleDomTree(tree)
22:    root ← tree.root()
23:    ruleset ← NewRuleset()
24:    ts ← NewTask(StyleNode(root, ruleset))
25:    tm ← NewTask(MatchNode(root, ruleset, ts))
26:    tm.SetSuccessor(ts)
27:    tm.Spawn()
28:    ts.Spawn()
29:    WaitForTasks()
30: end function
```

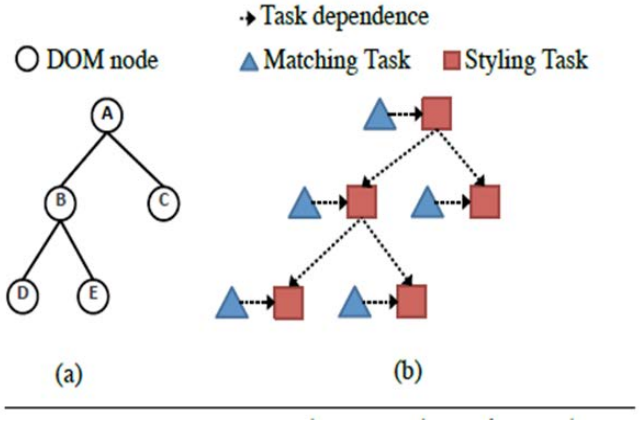

Figure. 4. DOM tree (a) and corresponding task DAG (b).

Algorithm 2 : Energy-Aware Approach

Begin to open a webpage
Data transmission computation is done.
Layout computation is finished
Collect features $\mathbf{x} = \{x_1, \cdots, x_{10}\}$
Webpage is opened
Wait for α seconds.
Get T_r from the prediction model with $\mathbf{x}$
if $(T_r > T_d)$ OR $(T_r > T_p$ AND mode == power) **then**
  switch to IDLE state
end if

We implement state switch functions at the application layer through RIL. To access radio firmware, the application sends a message which is an abstract description of an operation to be performed, to the RIL.java file in the application framework. Based on the received message, RIL.java sends another message to RIL through a UNIX socket connection. Then RIL operates on radio firmware to release the signaling connection and switch the smartphone to IDLE [1].

4.3 Network Capacity Analysis

When a user (smartphone) sets up a data session, the backbone network allocates a pair of dedicated transmission channels to the smartphone for downlink and uplink data transmissions, and then the smartphone enters the DCH state. The allocated transmission channels will be released after the smartphone leaves the DCH state. Due to the limited number of transmission channels, the backbone network may not always have dedicated transmission channels for a user's data session. With our energy-aware approach, smartphones will exist the DCH state quickly, and thus the dedicated transmission channels can be quickly released and being used for supporting more users. To quantify the benefit of our approach for supporting more users, we use network capacity, which is defined as the number of users that can be supported by the backbone network with certain quality, i.e., the session dropping rate (blocking rate) is below a pre-defined threshold β. The problem can be modeled as a M/G/N multi-server queue. Assume there are k users in a cell, and each user generates session requests following Poisson distribution and the average request rate of all users is λ. On the network side, there are N pairs of dedicated transmission channels in a cell, and it takes t time in average for a session to hold the channel. Here t is the whole webpage downloading time in the original approach, but it is much shorter in our energy aware approach.

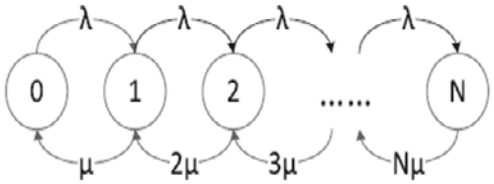

Figure. 5. The M/G/N queue model. User request comes at rate λ and the backbone network handles a request at rate μ .There are N pairs of dedicated channels [3].

Then, the average service rate is μ=1/t, and the network load is $\rho = \lambda/\mu$.
The M/G/N queue can be described as a markov chain as shown in Fig. 5. The nth circle (state) indicates the state with n sessions occupying the dedicated transmission channels, which have a probability of p_n. Each link indicates a transition, with the probability marked. According to the property of markov chain, we have:

$$\mu p_1 = \lambda p_0, \qquad p_1 = \rho p_0$$
$$2\mu p_2 = \lambda p_1, \qquad p_2 = \tfrac{1}{2}\rho^2 p_0$$
$$\ldots$$
$$n\mu p_n = \lambda p_{n-1}, \qquad p_n = \tfrac{1}{n!}\rho^n p_0.$$

Since the probability of all states is 1, we have n, and thus

$$p_0 = 1/\sum_{n=0}^{N} \frac{\rho^n}{n!}.$$

After N sessions have got the dedicated channels, later requests will be dropped. So the dropping rate equals to the state probability when there are N sessions in the system,

$$p_N = \frac{\rho^n / N!}{\sum_{n=0}^{N} \frac{\rho^n}{n!}}.$$

Then, the network capacity is the maximum number of users with dropping rate smaller than b, as shown in below:

$$k = \arg \min_{k} p_N < \beta.$$

5 Conclusion

In this paper, we proposed an energy efficient approach for web browsing in smartphones. First, we analyze the energy consumption of the Android browser at popular web sites such as Facebook, Amazon, and many others. This not only saves power, but also reduces the webpage loading time by removing the computation intensive redraws and reflows. Second, , to measure the energy needed to render individual web elements, such as cascade style sheets (CSS), JavaScript, HTML, images, and plug-in objects and also how to design web pages so as to minimize the energy needed to render the page .That results will shows in the graph and table format. During the session, web elements such as CSS and images will be cached locally. Therefore, we cannot estimate the energy cost of a session by simply summing the energies of pages visited during the session. Measuring an entire typical session may help optimize the power signature of the entire web site.

As future work, we will apply other techniques such as caching and prefetching to save energy. By prefetching and caching future useful data at the right time, wireless data transmissions can be aggregated to save energy. We will also promote our techniques to 4G/LTE network after it is widely deployed in our area.

Acknowledgments

We would like to thank Mr.Suhas Patil Assistant professor Computer engineering department, in K.J College of engineering and management research, Pune, Maharashtra, India for their useful guidelines.

References

[1] Bo Zhao, Wenjie Hu, Qiang Zheng, and Guohong Cao, —"Energy-Aware Web Browsing on Smartphones", IEEE Transactions on Parallel and Distributed Systems, 2014.

[2] Narendran Thiagarajan, Gaurav Aggarwal, Angela Nicoara, —"Who Killed My Battery: Analyzing Mobile Browser Energy Consumption", WWW 2012 – Session: Mobile Web Performance, April 16–20, 2012, Lyon, France, PP No. 41-50.B. Zhao, B. C. Tak, and G. Cao, "Reducing the delay and power consumption of web browsing on smartphones in 3G networks," IEEE ICDCS, 2011.

[3] Calin Cascaval, Seth Fowler Pablo Montesinos Ortego, Wayne Piekarski , Mehrdad Reshadi, Behnam Robatmili, Michael Weber, Vrajesh Bhavsar, "ZOOMM: A Parallel Web Browser Engine for Multicore Mobile Devices",Qualcomm Research Silicon Valleyzoomm@qualcomm.com.

[4] C. J. Kaufman, Rocky Mountain Research Laboratories, Boulder, Colo., personal communica K. Zhang, L. Wang, A. Pan, and B. B. Zhu, "Smart caching for web browsers", Int'l Conf. on World wide web (WWWW), 2010.

[5] Ekhiotz Jon Vergara, Joseba Sanjuan, Simin Nadjm-Tehrani Department of Computer and Information Science Link¨oping University, "Kernel Level Energy-Efficient 3G Background Traffic Shaper for Android smartphones", Sweden fekhiotz.vergara, joseba.sanjuan, simin.nadjm-tehranig@liu.se .

[6] B. Zhao, B. C. Tak, and G. Cao, "Reducing the delay and power consumption of web browsing on smartphones in 3G networks," in Proc. IEEE 31st Int. Conf. Distrib. Comput. Syst., 2011, pp. 413–422.

[7] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan, —"Bartendr: a practical approach to energy-aware cellular data scheduling", in Proc. ACM MobiSys, 2010.

[8] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study And implications for network applications," in Proc. ACM 9th SIGCOMM Conf. Internet Meas. Conf., 2009, pp. 280–293.

[9] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, —"A Close Examination of Performance and Power Characteristics of 4G LTE Networks", in Proc. ACM MobiSys, 2012.L. Hubert and P. Arabie, "Comparing Partitions," J. Classification, vol. 2, no. 4, pp. 193-218, Apr. 1985.

[10] W. Hu and G. Cao, "Energy optimization through traffic aggregation in wireless networks," in Proc. IEEE 33rd Annu. Int. Conf. Comput. Comm., 2014, pp. 916–924.

[11] J. Huang, F. Qian, Z. M. Mao, S. Sen, and O. Spatscheck, "Screen-off Traffic Characterization and Optimization in 3G/4G Networks," in Proceedings of the 2012 ACM Conference on Internet Measurement Conference, ser. IMC '12. ACM, 2012, pp. 357–364.