GreenBrowsing

Gonçalo Avelar and Luís Veiga
goncalo.avelar@ist.utl.pt, luis.veiga@inesc-id.pt

Técnico - ULisboa
INESC-ID Lisboa
Portugal

Abstract. Web 2.0 allowed for the enhancement and revamp of web pages aesthetics and interaction mechanics, leading to increasing energetic impact, proportional to the rate of appearance of more sophisticated browser mechanisms and web content. In this work GreenBrowsing is presented. This system is composed of (i) a Google Chrome extension that manages browser resource usage and, indirectly, energy impact by employing resource limiting mechanisms on browser tabs and (ii) a Certification sub-system, that ranks URL and web domains based on web-page induced energy consumption. We show that GreenBrowsing's mechanisms can achieve substantial resource reduction, in terms of energy-inducing resource metrics like CPU usage, memory usage and variation. This, with limited degradation of user-experience when compared to browsing the web without the extension.

Keywords: web browser, energy efficiency, web page certification, computational resource consumption

1 Introduction

As computing systems evolve, the energy spent in the provisioning of IT services increases. The carbon footprint of IT machinery becomes more evident and the energy costs of IT keep rising. As of 2008, the estimate was for *each single computer* in use to be capable of generating approximately a ton of carbon dioxide, yearly [5]. The trend is for the volume of emissions to continue to grow.

Considering the Web, it can be observed that the creation of more capable technologies (HTML5, CSS, JavaScript) improved connectivity and content delivery in the last few years, making website contents (sent to web browsers) substantially more resource-intensive. This ultimately leads to increased power consumption, since it seems reasonable to assume that *power consumption increases with resource usage*. Users should be aware of the power cost induced by visiting each web page, locally and remotely, relative to the power cost of other web pages, allowing them to choose between two or more functionally equivalent set of web pages the least power hungry.

Hence, we consider web browsers as suitable candidates for the deployment of a power management solution, and present GreenBrowsing: a system comprised of a (i) Google Chrome Extension, for the purpose of reducing energy consumption due to browsing the web and a (ii) Certification Back End, that ranks web-pages' domains and URLs according to their power-intensiveness.

The main challenge of this work was to provide a suite of mechanisms that effectively reduces the energy costs of web-browsing, without sacrificing much of the responsiveness and performance that is expected, and (at the same time) to provide means to certify web pages energetically-wise, in order to inform users of the what web-pages seem more power-consuming.

[1] The work presented in this dissertation was partially described in the following publication: Gonçalo Avelar, Luís Veiga. GreenBrowsing: Towards Energy Efficiency in Browsing Experience. DAIS 2014, LNCS, Springer (CORE B).

2 Design

There are two major subsystems that comprise GreenBrowsing: a Browser Extension that will act as a power manager, limiting browser access to resources (Section 2.1), and a Web Page Certification Back End, to be deployed as a prototypical big data analytics system (Section 2.2).

2.1 Browser Extension

The main roles of the Chrome's Browser Extension are *to reduce the resource consumption of idle tabs* (i.e. tabs not visualized by the user), and *to send to the Analytics Back End resource-consumption records*, used to derive energy consumption data, in order to certify web pages in terms of their energy consumption while being accessed.

In order to assess what are the resource consumption/user-perceived delays trade-offs involved in managing browser entities resource usage, different resource-limiting mechanisms will be presented. This will provide a varied mechanism catalogue to evaluate later, assessing what are the most prominent in reducing resource consumption rates while not impacting user experience, greatly.

Since Chrome employs a multi-process model, in which it might keep one process executing on behalf of one or more tabs, GreenBrowsing has to act directly upon the process responsible for handling each tab. This allowed to take advantage of some Operating System's capabilities, but also implies that some of the mechanisms considered will be OS-dependant.

The pseudo-code at Algorithm 1 summarizes the extension's behaviour for managing idle tab resource consumption. There is an initial test where, if a certain browser window is not focused (i.e. the topmost user-viewed window) all of the processes that handle its tabs will be halted. In other words, they will stop executing. On the other hand, if a certain window is focused, each of its tabs will be acted upon, individually. Firstly, if a certain tab is active (i.e selected by the user) it can consume as many resources it needs. If a tab is not active, the maximum resources its process is allowed to use will be limited. If that tab's process ever reaches the limits imposed, a certain *effect*/action will be cast upon that process. Both the resource type (e.g. CPU usage) and the expected effects on resource limit violation are mechanism-dependent.

```
Data: Windows
Data: Tabs
foreach window in Windows do
    if window is focused then
        foreach tab in Tabs do
            if tab not active then
                compute tab resource usage allowance ;
                apply resource consumption reduction mechanism ;
            else
                give unconditional resource consumption allowance to tab ;
            end
        end
    else
        foreach tab in Tabs do
            halt tab's process ;
        end
    end
end
```

Algorithm 1: Tab management algorithm overview.

The following formula is used to set the maximum resource usage (for any resource type), by taking into account the distance each idle tab is from the currently visualized tab, at a given moment, and the last time a certain idle tab was selected. Considering i as the tab index-distance from a certain tab to the active tab, within a certain window, p as the least-recently-used index relative to tabs within that same window, a as a controllable/user-defined *aggressiveness* exponent to further intensify reductions, if need be, and where $p >= 1$, $i >= 1$, $a >= 0$:

$$resource\_usage\_factor(i, p, a) = \frac{1}{p \ \times \ i^a} \tag{1}$$

The available GreenBrowsing mechanisms used to reduce resource consumption are presented as follows:

1. **Process Priority Adjustment (prio):** If there are x adjustable process scheduling priorities, ascendantly ordered by scheduling weight (where a value of x represents the most prioritary value and a value of 1 represents the least), the resulting priority of a certain tab's process will be given by:

$$round(resource\_usage\_factor(i, p, a) \times x) \tag{2}$$

The maximum value x for priority will be the one that represents a standard/normal priority given on process creation, by the operating system scheduler. Regarding the effect on resource limit violation, there is no concrete action taken. The only expectation is for a tab's process to execute less often relative to other processes (browser or any other application's related).

2. **Process CPU Rate Adjustment (cpu):** The rate adjustment will be a value in $[0, 100]$, where 0 represents no process usage allowed, and 100 means the process may fully utilize the processor, hence the adjustment will be computed as:

$$round(resource\_usage\_factor(i, p, a) \times 100) \tag{3}$$

If **cpu** is active, once a tab's process CPU usage reaches the limit set for that process, its execution is postponed, running again later, when it is given the chance to do so, by the Operating System's scheduler.

3. **Process Memory Limitation (mem):** With this mechanism, the maximum memory allowed for a process will be the maximum committed private memory up to the time that this mechanism was enforced. The adjusted memory value will be given by:

$$round(resource\_usage\_factor(i, p, a) \times max\_memory\_committed) \tag{4}$$

For **mem** there are two versions of this mechanism, with two different possible effect outcomes, once a memory limit is reached by a process: a (i) Soft version: the process is halted, and put to a sleep state, returning to execute once its tab becomes active, or a (ii) Hard version: the process is terminated, releasing all the resources allocated by it, until then.

4. **Process Execution Time Limitation (time):** In order to limit the duration a certain tab's process is allowed to run for, the average time between consecutive tab activations will be considered. The resource directly managed with this mechanism is execution time. The adjustment formula for allowed process execution time is compute as:

$$round(resource\_usage\_factor(i, p, a) \times average\_tab\_activation\_time) \tag{5}$$

For **time**, the effects employed on limit-breaching processes are the same as with *mem*, once the time for a tab's process to execute expires. It will also wield a Soft and a Hard version.

2.2 Back End

The Certification Back End Sub-System has the objective of *providing a clear and meaningful notion of how much energy web pages consume*. It is composed of three main components:

– A Data Store, where resource consumption records are stored together with the information needed to certify web-page URLs and domains;
– A Certification Server, that receives user-recorded resource consumption data, storing them at the Data Store. Its other role is to certify web-pages' URLs and domains.
– A Certification Modeller, responsible for devising the information needed to certify URLs and domains;

For each page, the resource metrics sent to the Back End's Certification Server will be (i) CPU usage (in terms of completed clock cycles), (ii) Private (main-)memory usage of processes (in Mega-Bytes), (iii) Network interface usage (in terms of the bits-per-second), to process and maintain each page open. These metrics were chosen because they were proved to be highly related to power consumption, in different settings ([8], [2], [6]).

```
**Input**: A set $O = \{O_1, O_2, \ldots, O_n\}$ of resource consumption values
**Input**: A set $C = \{C_1, C_2, \ldots, C_k\}$ of clusters' centers of mass
**Output**: A pair $\langle s, k\rangle$, where $s \in \{1, k\}$
$S \leftarrow \{S_1, S_2\}$
for $i \leftarrow 1$ to $n$ do
    $min \leftarrow -\infty$
    $\alpha \leftarrow k$
    for $j \leftarrow 1$ to $k$ do
        $distance \leftarrow d(O_i, C_j)$
        if $distance < min$ then
            $min \leftarrow distance$
            $\alpha \leftarrow j$
        end
    end
    $S_\alpha \leftarrow S_\alpha + 1$
end
$s \leftarrow i$, where $S_i > S_j, \forall\langle S_i, S_j\rangle \in S$
return $\langle s, k\rangle$
```

Algorithm 2: Certification Algorithm used to score web-page URL and domains.

The Back End is also responsible for devising certification categories/ranks to be associated with web-page URLs and domains, on the act of certification. This at the Certification Modeller by employing a method know as *Expectation-Maximization* [4]. The basic idea is to cluster the observations recorded into, no less than, 8 categories. Two different data sets will be used to compute parameters for two different models – one comprising resource usage associated with URL and another for web-page domains, being the URL dataset contained in the domain dataset. The observations belonging to the multivariate resource consumption random variables are assumed to be normally distributed, so Multivariate Gaussian Mixture Models are used to fit the data and to iteratively train clusters. Once they are trained, a random selection of observations is *sampled* from them. Each sample represents its cluster through the sample's Center of Mass/Centroid. This Center of Mass is the information used when certifying pages' URLs and domains, at the Certification Server. The certification is formally presented in Algorithm 2 (where k represents the number of certification categories, n the total number of observations).

3 Implementation

3.1 Browser Extension

The Browser Extension was implemented focusing Chrome's deployment on the Windows Operating System. Since Chrome has very limited support for process management, namely of its tabs, the Extension needed to be divided in two main entities: (i) **The Browser Extension** itself, comprised of JavaScript callbacks and code rather event-oriented, whose execution and handling is delegated to the Browser, by running from within the Browser itself as a Google Chrome Extension. (ii) A **Background Process (BP)** running natively as a Windows application. Through it, browser processes can be directly managed by communicating, beforehand, with the extension.

When on Windows, Chrome uses Windows Job Objects to employ part of its sandboxing constraints. Job Objects are Windows abstractions that allow the grouping of processes and the enforcement of certain limits and restrictions over them. This is exactly what is needed in order to implement the resource limiting mechanisms described at Section 2.1.

The Sandboxing Model used by Chrome prescribes the association of a single tab process to a single Job. Knowing that these Job objects are kept at Chrome's Kernel Process – i.e. the process that orchestrates all browser activity, from tab creation and management to resource access – the BP retrieves these jobs by enumerating all the Windows Kernel Objects present at Chrome's Kernel Process, keeping those that correspond to Job Objects. Once all Job Objects are found, the association of jobs to tab processes is done by calling the WinAPI fumction `IsProcessInJob`. Tab processes are retrieved by querying the browser, through its JavaScript API. This is done at the Browser Extension which, in turn, will pass the tab-to-process associations to the BP, where they are associated with Jobs.

The Tab Management Algorithm described at Section 2.1 will therefore limit resource usage by acting directly on Jobs. One detail about *soft* mechanisms is that processes can be halted by removing each of its threads from the execution queue of the Operating System scheduler. Limit violations are periodically checked by a listener thread, that is responsible for halting tab processes, in case they breached their allowed resource consumption limit.

3.2 Certification Back End

Concerning the Back End subsystem, all code was developed on Java. Communication between components is done via JSON over TCP.

The Certification Server utilizes the a netty-extended socket.io framework, to serve incoming certification requests. This framework is an implementation of the WebSocket protocol and allows to serve requests efficiently and asynchronously.

The Certification Modeller runs as a process with two Java threads. Each thread computes the model used to certify either URLs or domains. This is done using a combination of Apache Spark built-in Expectation Maximization function, for Multivariate Gaussian Mixtures and Apache Commons Math library, for the sampling of clusters.

For storing resource consumption records, coming from the Certification Server, and the model's centers of mass, coming from the Certification Modeller, a PostgresSQL database is deployed at the Data Store.

4 Evaluation

Tests were scripted combining sequences of *mechanisms* with *aggressiveness* values of 1 and 1024. A set of typical web-pages was used, comprising pages of news sites, social networks, sports sites, mail clients and multimedia-streaming sites, providing a varied web-page suite. Scripts, firstly, open a set of pages and, secondly, navigate through those pages, gathering resource consumption data, while pages are navigated. Every time a tab is terminated, due to employing *mem hard* or *time hard*, it has its page reloaded once it becomes active again. Three *tab selection policies*, stating what is the next tab to activate (i.e. what page to visualize next), were used: (i) *round-robin selection* to navigate sequentially from tab to tab; (ii) *central tab incidence*, where the tabs at the center of the tab bar will be selected more often, by following a periodic navigation scheme, from the first tab to the last and from the last to the first one, in a back and forth-fashion; (iii) *random tab selection* where a certain tab is selected randomly, possibly more than once. Regarding the testing environment Chrome version was 44.0.2391.0, dev-channel release. The operating system on which Chrome was installed was Windows 8.1 Pro – baseline install, no updates. Hardware-wise, the tests were conducted on a ASUS K50IN laptop, Intel®Core(TM)2 Duo CPU P8700 running at 2.53GHz, with 4GB of RAM memory.

4.1 Resource Usage Evaluation

The resource variations induced by *prio* might not noticeable do to the naked eye because of the highly variable values of CPU usage rates, over time. Reductions of 9.92% and 17.56% were recorded, however, being the latter recorded with an higher value of aggressiveness, as shown in Figure 1, in green.

When applying *cpu* (Figure 2), the reductions in CPU usage are intensified even more when compared with *prio*, this time holding reductions that range from 20% to about 47% of CPU time. This seemingly advantage over *prio* was expected, since *cpu* directly adjusts the CPU usage allowed for each tab's process, contrary to *prio*, that associates priorities to a process without adjusting the maximum value for CPU usage, itself.

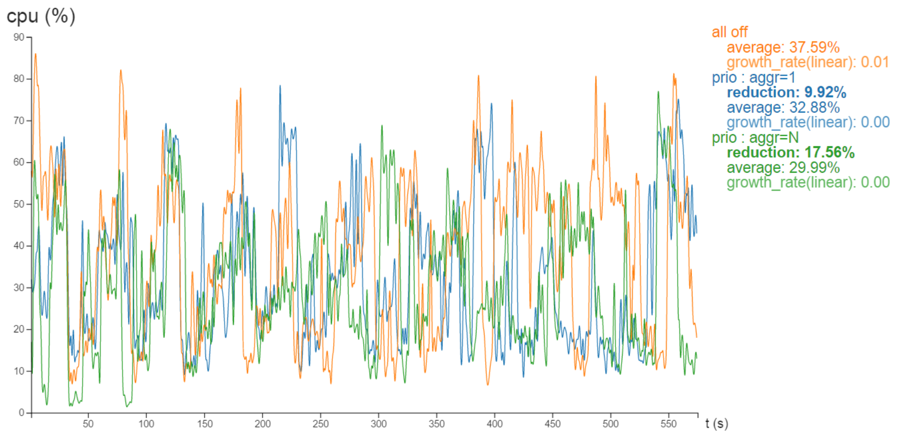


Fig. 1. CPU usage for *prio*.

Figures 3 and 4 depict how applying *mem soft* and *time soft* influenced CPU usage. The first seems to be the most prominent in reducing CPU usage, with 80% reductions, while the latter is still successful in doing so, even though to a lesser extent, achieving close to 70% reductions.

Concerning memory usage (Figures 5 and 6), *hard* mechanisms induce a substantially lower memory usage, than their *soft* counterparts, achieving reductions of 80% to 85%, when compared to mechanisms being *all off*.

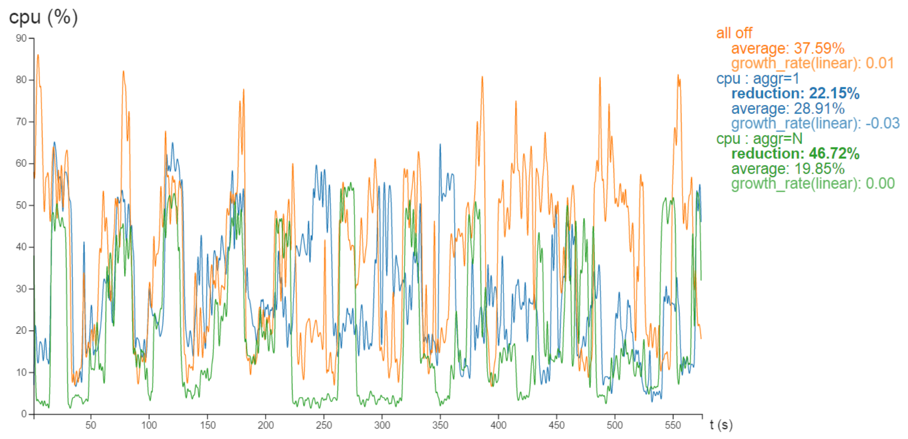


Fig. 2. CPU usage for *prio* & *cpu*.

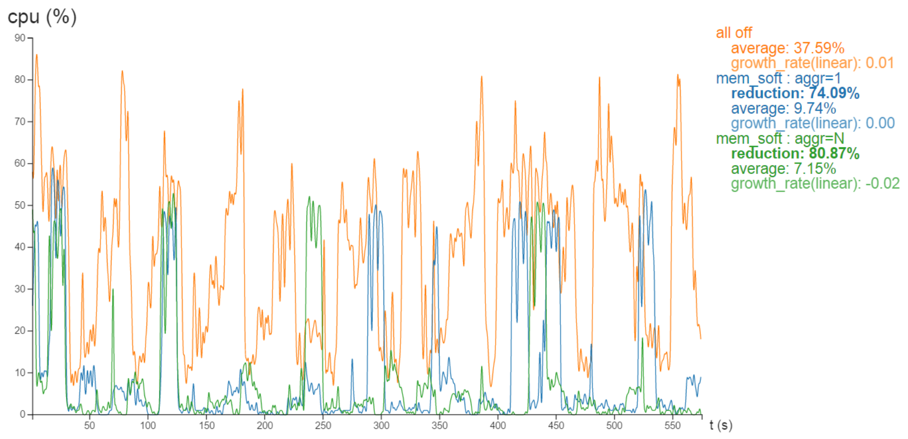


Fig. 3. CPU usage for *mem soft*.

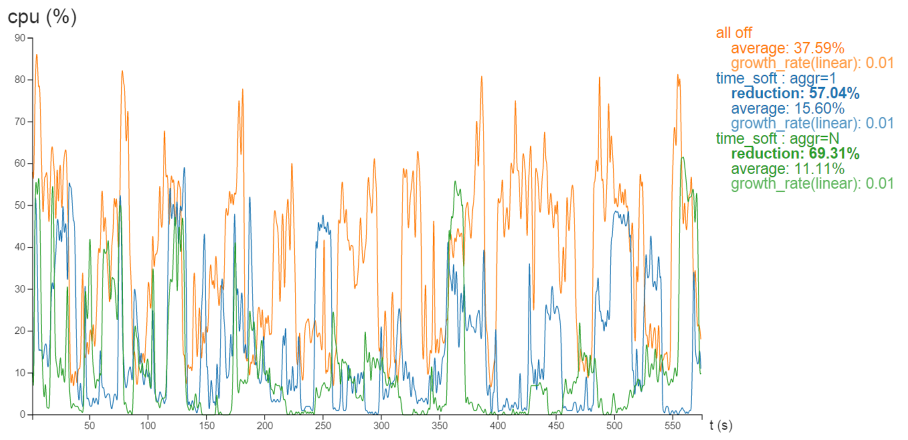


Fig. 4. CPU usage for *time soft*.

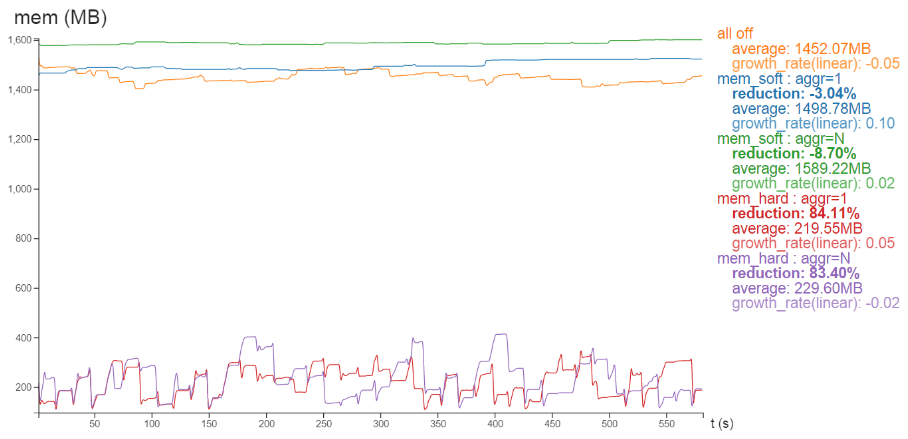


Fig. 5. Memory usage for *mem soft* & *mem hard*.

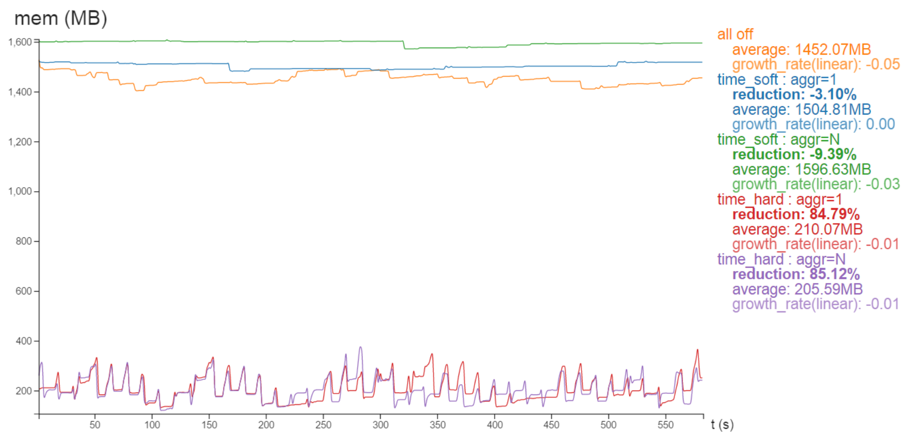


Fig. 6. Memory usage for *mem soft* & *mem hard*.

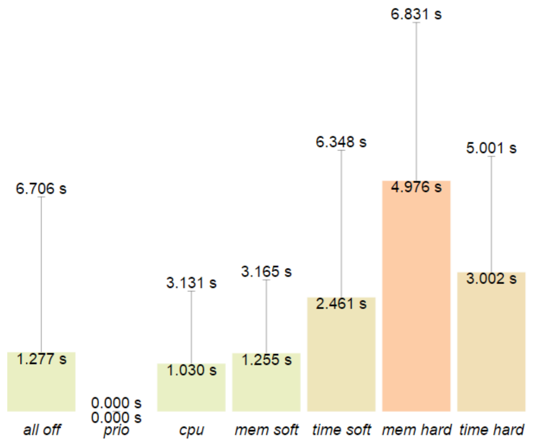
Overall, *mem soft* and *time soft* seemed to be the most capable mechanisms, in terms of managing idle tab resource consumption. Its reductions beat all of other mechanisms, when it comes to CPU usage. Even though experiments at Figures 5 and 6 seem to disprove its effectiveness in reducing memory usage, (since *soft* mechanisms achieved slight increases when compared to *all off*), it is important to notice how stable memory consumption was when compared to the memory variations induced by other mechanisms and *all off*, over time. If it is assumed that memory variations represent system-wide activity, due to having many system entities accessing it, and therefore inducing energy consumption rates proportional to the variations recorded, then *soft* mechanisms effectively help reduce energy consumption, by varying the least.

4.2 Perceived Delays Evaluation

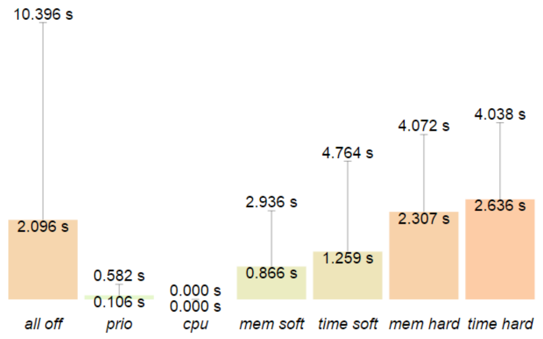
In order to assess what are the implications in terms of user experience-significant requirements, Latency was recorded, while running resource consumption tests. Latency, in this context, corresponds to the time period that goes from the moment the active tab starts loading web-page content to the moment that content is totally loaded. This notion of latency is useful to give an idea of how much time is wasted, by enforcing certain mechanisms, in comparison to others.

Figure 7 presents the latencies experienced on average, as rectangles, for each tab selection policy. Standard deviations correspond to the vertical lines above rectangles. It is possible to see that latencies for *hard* mechanisms were always bigger, on average, when compared to other mechanisms. The experiments comprising *all off*, *prio* and *cpu* held the smaller latency values, as expected, since they tamper very little with process functioning, when compared to other mechanisms (namely the *soft* and *hard* ones). It is possible to observe that *soft* mechanisms seem to achieve acceptable latencies, when compared to *all off*. The exception is when tabs were chosen randomly, where the latency values are comparable to those recorded for *hard* processes. The standard deviations observed are rather high in value. It has to do with the wide latency-value-ranges recorded since, occasionally, some long periods of consecutive busy-tab activations were recorded (where the activated tabs were still processing their pages).

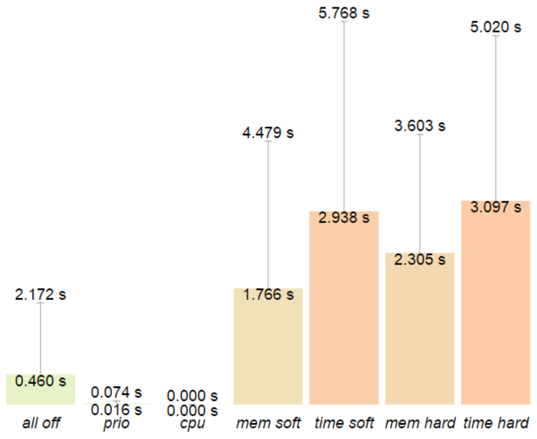
It seems, therefore, negotiable to apply all mechanisms for resource reduction purposes, with the exception of *hard* mechanisms, given the latencies recorded for them, in most experiments.



(a) Latency measurement / round-robin tab selection



(b) Latency measurement / central-tab-incidence selection



(c) Latency measurement / random tab selection

Fig. 7. Latency measurements for the 3 tab selection policies considered.

5 Related Work

The problem of achieving low energy consumption rates, on computational systems, has been defined by Benini et al. as Dynamic Power Management (DPM) [1].

Dynamic Power Management techniques try to achieve power dissipation reductions by employing policies that reduce the performance of system components (typically by inducing them into some sort of sleep state) when they are idle, while under performance constraints.

One pioneering example of DPM is the work by Qiu et al. [7], that follows a stochastic approach. The authors describe a continuous-time Markov Decision Process (MDP) as the system's power model. Each state transition has a score assigned to it, consisting of the product of the probability of that transition occurring, the energy cost of that transition and a certain weight. The idea is to minimize the sum of all scores, over periods of time, by iteratively adjusting these weights.

In terms of Energy-related Certification, there is some work targeting different kinds of computational systems. Camps et al., for one, proposed a solution to do so [3], however the certification is done accounting only for the downloadable content of pages.

6 Conclusions & Future Work

In this work, GreenBrowsing was presented as a system comprised of (i) a Google Chrome Extension, used to reduce consumption of idle tabs, being the Extension developed for Chrome running over Windows, and (ii) a Back End subsystem, that certifies URLs' and domains' webpages energetically-wise, in regard to the resource consumption introduced in processing them. Evaluation showed that substantial resource usage might be decreased with acceptable impact on user-perceived delays, when comparing various GreenBrowsing mechanisms' operation with no mechanism employment at all.

Regarding future work, more resource reduction mechanisms could be devised in order to account for bandwidth usage, since studies show it plays a significant part on energy consumption, specially in the case of wi-fi enabled devices. The Back End would benefit from improvements at the Data Store, in order to improve its scalability when it comes to processing reads and writes of resource consumption records.

References

1. Benini, L., Bogliolo, A., Cavallucci, S., Riccó, B.: Monitoring system activity for os-directed dynamic power management. In: Proceedings of the 1998 International Symposium on Low Power Electronics and Design. ISLPED '98, ACM, New York, NY, USA (1998)
2. Bircher, W.L., John, L.K.: Complete system power estimation using processor performance events. IEEE Transactions on Computers 61(4), 563–577 (2012)
3. Camps, F.: Web browser energy consumption (2010)
4. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. Journal of The Royal Statistical Society, Series B 39(1), 1–38 (1977)
5. Murugesan, S.: Harnessing green it: Principles and practices. IT Professional 10 (2008)
6. Park, J., Yoo, S., Lee, S., Park, C.: Power modeling of solid state disk for dynamic power management policy design in embedded systems. In: Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems. pp. 24–35. SEUS '09, Springer-Verlag, Berlin, Heidelberg (2009)
7. Qiu, Q., Pedram, M.: Dynamic power management based on continuous-time markov decision processes. In: Proceedings of the 36th Annual ACM/IEEE Design Automation Conference. DAC '99, ACM, New York, NY, USA (1999)
8. Rodrigues, R. Koren, I.K.S.: A study on the use of performance counters to estimate power in microprocessors. In: Circuits and Systems II: Express Briefs, IEEE Transactions (2013)