TÉCNICO LISBOA

GreenBrowsing

Gonçalo João Curado Avelar

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Professor Luís Manuel Antunes Veiga

Examination Committee

Chairperson:	Professor José Carlos Alves Pereira Monteiro
Supervisor:	Professor Luís Manuel Antunes Veiga
Member of the Committee:	Professor Rui António dos Santos Cruz

May 2015

Dedicated to my family.

Acknowledgments

I would like to thank Professor Luís Veiga for all the support, guidance and patience provided while supervising me. I am, also, grateful for the opportunity to experience, first hand, what is like to attend and present my work at a scientific conference. Finally, I thank him the mixture of motivational, enthusiastic and scientific-oriented psyche he passed me.

I thank my family for the unconditional support and encouragement provided in ill-fortuned situations. I also thank the cheering when successfully accomplishing objectives.

Resumo

O fenómeno web 2.0 permitiu o melhoramento das páginas web, em termos de estética e interatividade. No entanto, estes melhoramentos levam a um impacto energético acrescido, proporcional ao aparecimento de mecanismos e conteúdos mais sofisticados. Neste trabalho, o sistema GreenBrowsing é apresentado. Este é composto por (i) uma extensão do browser Google Chrome, responsável por gerir o impacto nos recursos e, indirectamente, energético inerente ao acto de navegar na web, aplicando mecanismos que limitam o acesso de tabs a recursos computacionais. É também composto por (ii) um sub-sistema, Back End, responsável por certificar o URL e domínios, em termos do impacto energético devido ao processamento das respectivas páginas web. A avaliação experimental mostra que o sistema GreenBrowsing é capaz de reduzir, substancialmente, o consumo de recursos computacionais, em termos de métricas directamente relacionadas com consumo energético. Exemplos são a utilização de CPU, bem como a utilização e variação de acessos a memória virtual. A utilização da largura de banda é, também, indirectamente reduzida através da aplicação de certo sub-conjunto dos mecanismos desenvolvidos. A avaliação mostra também que a degradação da experiência do utilizador é limitada, quando comparada com o navegar na web sem a acção da extensão.

Palavras-chave: web browser, eficiência energética, certificação de páginas web, consumo de recursos computacionais

Abstract

Web 2.0 allowed for the enhancement and revamp of web pages aesthetics and interaction mechanics. Unfortunately, this leads 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. It is also, indirectly and partially, able to reduce bandwidth usage when employing a specific subset of the mechanisms presented. All 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

Contents

Acknowledgments v
Resumo vii
Abstract ix
List of Tables xiii
List of Figures xvi
Acronyms xviii

1 Introduction **1**
1.1 Challenges and Solution Requirements 2
1.2 Shortcoming of Current Solutions 2
1.3 Contributions 3
1.4 Document Structure 3
1.5 Publications 4

2 Related Work **6**
2.1 Dynamic Power Management 6
2.1.1 Architectural Overview. 7
2.1.2 Classification of Dynamic Power Management Systems. 7
2.2 Energy-Aware Scheduling 14
2.2.1 Classical Scheduling Algorithms. 15
2.2.2 Reference Energy-Aware Scheduling Algorithms. 15
2.3 Energy-related Certification and Analytics on the Cloud 17
2.3.1 Energy-related Certification Computational Systems. 17
2.3.2 Classes of Big Data Analytics System. 18
2.3.3 Relevant Energy-related Big Data Analytics Systems. 20
2.4 Analysis and Discussion 22

3 Architecture **24**
3.1 Extension Sub-System 25
3.1.1 Terminology. 25
3.1.2 Modules and Run-Time Components Description 27
3.1.3 Tab Management Algorithm. 28

3.1.4 Mechanisms for Resource Reduction. . . . 30
3.2 Analytics & Certification Back End Sub-System . . . 33
3.2.1 Components of the Certification Sub-System. . . . 33
3.2.2 Performance Counters for Energy-related Certification. . . . 34
3.2.3 Requirements for Energy-related Certification. . . . 35
3.2.4 Devising Certification Categories. . . . 35
3.2.5 The Certification Algorithm. . . . 37
3.2.6 Resource Usage Gathering. . . . 37

4 Implementation **40**
4.1 Browser Extension . . . 40
4.1.1 Browser Extension & Background Process. . . . 40
4.1.2 Operating System Facilities. . . . 42
4.1.3 Soft Mechanisms. . . . 42
4.1.4 Hard Mechanisms. . . . 43
4.2 Back End . . . 44
4.3 Final Implementation Considerations . . . 46

5 Evaluation **48**
5.1 Experimental Setup . . . 48
5.2 Resource Consumption Analysis . . . 49
5.2.1 CPU Usage. . . . 50
5.2.2 Memory Usage. . . . 53
5.2.3 Network Bandwidth Usage. . . . 56
5.2.4 Discussion. . . . 59
5.3 User Experience Evaluation . . . 60
5.3.1 Frame-rate Accounting. . . . 60
5.3.2 Latencies per-Mechanism. . . . 61
5.3.3 Resource Consumption and Latency. . . . 62
5.4 Evaluation Summary & Conclusions . . . 66

6 Conclusions & Future Work **69**
6.1 Conclusions . . . 69
6.2 Future Work . . . 70

Bibliography **75**

List of Tables

2.1 Markov Model classification. 9
2.2 Markov Model Time Set and State Space cardinality classification. 9
2.3 Dynamic Power Management Schemes Classification. 14
2.4 Big Data System Features. 21
2.5 Big Data System Classification. 22

3.1 Tab management algorithm overview. 29
3.2 Mechanisms summarized classification. 32
3.3 Algorithm for modelling Certification Categories. 36
3.4 Certification Algorithm used to score web-page URL and domains. 37

4.1 Resource Consumption Table details. Row names in **bold** represent primary keys. 45
4.2 Model Parameters details. Row names in **bold** represent primary keys. 45

List of Figures

2.1 Dynamic Power Management Architecture 7

3.1 Symbolic depiction of GreenBrowsing actuating on idle tabs and energy-related web-page certification. 24
3.2 Disposition of Browser entities. All tabs are *complete*, with the exception of the right-most tab, which is loading. 26
3.3 Layered View of The Browser Extension. 28
3.4 Effects of tab management algorithm on tab resource consumption. 30
3.5 Certification requests sent from GreenBrowsing users to the Certification Server. 34
3.6 The Communicating tasks of the Back End. 34

4.1 Tab State Consistency Flowchart 41
4.2 Faulty effects on silent Process Termination 44
4.3 Certification Server Processes & Threads 45
4.4 Certification Modeller Processes & Threads 46

5.1 Mechanism prio / cpu usage 51
5.2 Mechanism cpu / cpu usage 51
5.3 Mechanism mem hard / cpu usage 52
5.4 Mechanism time hard / cpu usage 52
5.5 Mechanism mem soft / cpu usage 53
5.6 Mechanism time soft / cpu usage 53
5.7 Mechanism prio / memory usage 54
5.8 Mechanism prio / memory usage 54
5.9 Mechanism mem hard & mem soft / memory usage decreasing 55
5.10 Mechanism time hard & time soft / memory usage decreasing 55
5.11 Mechanism mem hard & mem soft / Memory usage stable 56
5.12 Mechanism time hard & time soft / Memory usage stable 56
5.13 Mechanism prio / bandwidth usage 57
5.14 Mechanism cpu / bandwidth usage 57
5.15 Mechanism mem hard & mem soft / Bandwidth usage 58
5.16 Mechanism time hard & time soft / Bandwidth usage 58

5.17 FPS measurements – Averages and Standard Deviations 61
5.18 Latency measurements – Averages and Standard Deviations 62
5.19 Resource Consumption and Latency Correlation when no mechanisms are active 64
5.20 CPU usage and latency correlation for mem 64
5.21 CPU usage and latency correlation for time 64
5.22 Memory variation and latency correlation for mem 65
5.23 Memory variation and latency correlation for time 65
5.24 Bandwidth usage and latency correlation for mem 66
5.25 Bandwidth usage and latency correlation for time 66

Acronyms

BP Background Process.

CM Center of Mass.

CPU Central Processing Unit.

DFVS Dynamic Frequency and Voltage scaling.

DPM Dynamic Power Management.

EM Expectation Maximization.

FCFS First-Come-First-Served.

FPS Frames-Per-Second.

HPC High Performance Computing.

IOCP I/O Completion Port.

LDLR Local-DataStore-Local-Runtime.

LDRR Local-DataStore-Remote-Runtime.

MDP Markov Decision Process.

MGMM Multivariate Gaussian Mixture Model.

OCA Observer-Controller-Adapter.

PM Power Manager.

QL Q-Learning.

RDRR Remote-DataStore-Remote-Runtime.

RL Reinforced Learning.

SA Simulated Annealing.

SLA Service Level Agreement.

SMDP Semi-Markov Decision Process.

SP Service-Provider.

SQ Service-Request-Queue.

SR Service-Requester.

SRSP Service-Requester-Service-Provider.

TD Temporal Difference Learning.

VCPU Virtual CPU.

VM Virtual Machine.

Chapter 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 [Murugesan, 2008]. The trend is for the volume of emissions to continue to grow.

In order to create more sustainable and energy efficient computing systems, measures must be taken regarding the ways systems are designed, used, manufactured and even disposed of [Murugesan, 2008]. This applies to servers, networking infrastructure components, (like switches and routers), and the devices the end-users resort to.

The means to reduce power consumption could be both hardware or software based. However, in the context of the World Wide Web and at the scope of end-user devices, the web browser should be one of the components to focus on, when it comes to power management and resource limitation.

Due to the improvements in connectivity and delivery of content in the last few years, it is now possible to share a lot more information than it, otherwise, would be. The Web 2.0 phenomenon lead to the creation of more capable technologies, (HTML5, CSS, JavaScript), powering blogging platforms, social networks, and multimedia-streaming sites. By being assembled with these technologies, website contents are sent to web browsers where they are processed and, more often than not, behave more like a true application than a static web page. As a result, the power consumption in a single end-user device, derived from web browsing, is two to three orders of magnitude larger than in all the intermediate routing equipment, found in the traversed network path [Gyarmati and Trinh, 2011]. This relation between the different machinery that operate on the Internet suggests much more could be done regarding the way web pages are processed and demanded by browsers. To that effect, two scenarios can be considered:

- either people start browsing the web more responsibly, requesting each page at a time, lowering the resource consumption on their devices, and therefore lowering power consumption rates, (which could be perceived as a loss of convenience and business value); or
- developers become more responsible for the software they develop, making energy-efficiency a primary requirement, taking it into account since they start developing their systems.

The first scenario is an improbable one. It is hard to instigate environmental responsibility and energy-awareness into users minds, mainly because the financial and energetic incentives, to make people adopt energy management strategies, are minor compared to the constant ”desire for always available computing” [Chetty et al., 2009]. In the same study, it is also suggested that ”people do not necessarily choose their automated power management settings”. Even though this was a study on energy inefficiencies derived from domestic computer usage, it is reasonable to assume that the same ideas hold in more specific cases like the one of web browsers. Another hint of the users indifference towards green software, can be found in a study by Amsel et al. [Amsel et al., 2011]. It seems that energy-awareness must be delegated to the developer, instead of the user. In fact, some already argued in favour of that [Miettinen and Nurminen, 2010]. One major concern in the previous claim is that it is not feasible to reconstruct existing web applications, mainly because of the programmatic effort it would require.

Therefore, what *power management strategies* should be employed in order to provide power consumption reductions, while browsing the web? How can environmentally concerned users, or even simple-minded users alike, be assured that certain web pages are *greener* than others? How can the related web page processing be used to instigate energy-awareness?

These questions motivate the solution proposed and appears as an alternative to the scenarios discussed previously, by extending the underlying runtime systems and application environments – web browsers – to monitor, promote and certify resource efficiency of running applications – web pages.

1.1 Challenges and Solution Requirements

The main challenge of this work was to provide mechanisms that effectively reduce the energy cost when browsing the web, without sacrificing much of the availability and performance that is expected, while browsing, and by providing means to certify web pages energetically-wise, in order to inform users of the energetic inefficiencies related to different web page visualizations.

Other more specific challenges encompassing the use of undocumented functions for accessing certain Operating System facilities, for one, needed some extra research, experimentation and prototyping. Only then, it was possible to deliver a final solution to the problems of reducing resource/energy consumption imposed.

1.2 Shortcoming of Current Solutions

Current solutions lack the context at which they were supposed to perform power management actions (the web browser runtime state). Moreover, they typically oversee components metrics, like CPU utilization, disregarding other important components like main memory, which are also responsible for a reasonable slice of the overall energetic waste [Bircher and John, 2012]. An example is Chameleon [Liu et al., 2005], that brings power management to the application level, adjusting the speed at which applications run. This might be bad design, since users often impose tight availability constraints on the

systems they use. Reducing application running speed might lead to negative user experience. On the other hand, there is ACE [Yan et al., 2005]. These systems try to leverage the web browsing networking behaviour, by reducing user-perceived page fetch latency, allowing also for energetic gains. However this focus solely on power management details of connections, disregarding other resource hungry components of a web browser, like the idle pages' local processing.

1.3 Contributions

Power consumption is proportional to computational resource usage.

This claim serves as the basis for this work and it encompasses a system that manages browser access to resources, through the enforcement of different mechanisms that limit resource usage.

The case study and implementation will be done on top of Google Chrome Web Browser [Reis and Gribble, 2009]. Chrome is a complete web browser. By complete it is meant to be more than hyper-text page retriever. It embodies a full application execution environment with JavaScript just-in-time compilation, garbage collection, thread and process management, and component-oriented architecture. In essence, a virtual machine for the web. Chrome also allows installation of extensions, that range from games to plug-ins with daemon-like behaviour, (for instance Addblock). Its selection has to do with studies showing that it is one of the most power consuming browsers and, in general, one of the most power consuming applications [Patel and Perkinson, 2013, Bianzino et al., 2011].

The version of Chrome targeted is deployed on the Windows Operating System. GreenBrowsing aims at extending Chrome, in order to decrease the energy costs of browsing, by taking into account idle tabs (tabs that are open but not being used), as well as taking advantage of the browser API to perform energy-related optimizations.

An energy-related web page certification scheme will also be presented, based on computational resource consumption, to the end of raising user awareness in regards to what pages are more resource hungry.

1.4 Document Structure

The Document is organized in the following manner:

In Chapter 2, both seminal and state-or-the-art energy-reduction systems will be surveyed. The idea is to draw the needed mindset and rationale from solutions to abstract problems (or generally browser-unrelated domains) and see how they could provide useful to the problem of reducing the energy usage inherent to web-browsing. Energy-related certification systems will also be studied in order to account for the best practises in gathering and analysing data, however to a lesser extent.

In Chapter 3, the architectural choices and the algorithms relative to this work will be described, for both the power management extension and the certification sub-system.

In Chapter 4, the implementation details will be explained, accounting for platform specific problems/dilemmas and how they were overcome.

In Chapter 5, the evaluation methodology will be presented, as well as the evaluation testing done in regards to the resource reduction achieved and user-perceived latency impact.

Finally, in Chapter 6, final remarks and conclusions will be done. Directions for future work will be given.

1.5 Publications

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).

Chapter 2

Related Work

The next sections will present the work and areas of research that are related to this context of energy-aware web browsing and were more relevant to our proposed work. Section 2.1 will provide a description of how to dynamically manage power consumption. Section 2.2 will present scheduling algorithms to reduce energy losses, in multi-task environments. Section 2.3 will address big data and energy analytic systems.

2.1 Dynamic Power Management

Many of the hardware devices and software components that belong to computational systems are event-driven. Events can be direct commands issued by other components or other kinds of events like I/O interruptions, but in essence they are asynchronous and lead to intermittent work periods. Between work periods, idle periods take place. However, during idle periods energy is still consumed. If the typical data-center scenario is considered, when server utilization is below 30%, idle servers still consume the equivalent to 60% of their work-peak power consumption [Meisner et al., 2009]. This asymmetry needs therefore to be dealt with.

Dynamic Power Management, (DPM in short), is the ability to reduce power dissipation, by selectively turning off, or reducing the performance of a system's components when they are idle, (or partially unexploited) [Paleologo et al., 1998]. These reductions of power dissipation are typically subject to performance and inherent quality of service constraints.

We will start by presenting, in a top-down approach, the characteristics of Dynamic Power Management systems. In Section 2.1.1 we will explain the architecture of dynamic power management, at the highest abstraction level possible. In Section 2.1.2 we will present a classification of DPM systems depending on the different aspects of the DPM architecture elements. Finally, in Section 2.1.2, some DPM solutions will be presented and classified according to the aspects described in Section 2.1.2.

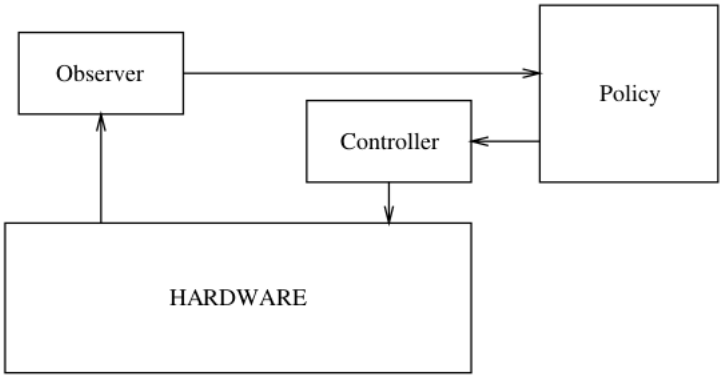


Figure 2.1: Dynamic Power Management Architecture. As seen in the work of Benini et al. [Benini et al., 1998]

2.1.1 Architectural Overview.

Benini et al. [Benini et al., 1998] establish a fundamental approach to system-level dynamic power management by providing an high-level architecture, composed by three main components: the Observer, the Controller and the Policy, (as seen in Figure 2.1). The latter takes power-management decisions. These decisions are based on the information gathered and transmitted by the Observer, as it monitors system activity. The Controller is the component through which power management decisions are enforced, on behalf of the Policy.

In practice, the Observer corresponds to the components that interact with the OS and other device APIs, gathering system properties like CPU and memory usage. The controller is the one who talks directly to devices through device drives. The Policy is the component responsible for making sense from the gathered data – by the Observer – and issue calls to the right system components – through the Controller.

2.1.2 Classification of Dynamic Power Management Systems.

The functioning of a Dynamic Power Management system is related to certain aspects that depend both on the relations between components and the system under management. The way that components interact and the way that power management decisions are enforced might penalize performance. In this Section we present a classification that emphasizes the performance penalties of different design choices for dynamic management systems.

There are some ubiquitous details that influence decision making. These details arise from the way the system under management behaves and reacts to the actions that are performed on it. To that end, policies should be based on proper Power and System models.

The decision criteria that allows for a certain system to be adjusted in terms of power consumption, with respect to a systems state change, is embodied in *Power Models*. Through the enforcement of

Power Models, the Policy can adapt to different workload scenarios, adjusting its decision making mechanisms, in order to perform better power management actions. In essence, Power Models provide a formal description of the conditions that need to be met, accounting for both system characteristics and other constraints, (like performance and availability).

Heuristic Power Models.

The more intuitive approach to provide some means of policy adaptation is through the establishment of a static set of rules. These rules are based on common system behaviour and can be implemented as functions, whose parameters correspond to observations and measurements gathered during system's execution. This is the essence of heuristic power models. When modelling simple systems, under near-always-right assumptions, these might suffice in providing good power management capabilities.

Stochastic Power Models.

A stochastic model [Klebaner, 2012] is one that is based on the notion of stochastic process: set of *random variables X(t)*, as a function of time *t*, whose values are called states, and the set of possible values is called state space. In this way, a stochastic model models a process where the current system's state depends on previous states in a non-deterministic way.

Stochastic models try to solve the problem of Dynamic Power Management in a different way of Heuristic Models. They try to answer the following question: Given the current state of the system, what better (future) sequence of actions could minimize the power consumption, knowing that the system can change amongst different power modes at any given moment? By calculating the probabilities of different sequence of actions and by weighting together with performance costs of applying each action and transiting from one state to another, (among other variables), stochastic models are used to generate policies that execute well under specific performance constraints (but still Heuristic models typically perform better). The problem however is to devise optimal policies, since systems often behave unpredictably.

Amongst the many types of stochastic models, are the widely used Markov Models [Kaelbling et al., 1998]. In these models the Markov Property [Norris, 1998] holds, hence their name. Intuitively, the Markov Property tells us that given a sequence of N events, the value of the probability of the n^{th} event happening after some exact sequence of $N-1$ previously observed events is approximately equal to the value of the probability of the n^{th} event happening after the $n-1^{\text{th}}$. This approximation is quite handy, since it just requires the computation of the probability of a certain event n^{th}, conditioned to the previous $n-1^{\text{th}}$ one, disregarding all the events observed previously.

Table 2.1 shows how Markov Models can be classified regarding *observability* of system events and *control* over the system where the Markov Model is applied. In a *partially observable system*, not all states are known beforehand, being discovered dynamically. In a *controlled system*, the Markov Model state transitions depend on the current state and on an action that is applied to the system. Therefore each state is associated to a certain action. In the context of DPM, it means that when the system is in a

certain state, the Policy will perform the corresponding action over some power consuming components. Of course, to that effect, there must be some sort of relation amongst the state set and the components under management, by the Policy.

	Observable System	Partially Observable System
Autonomous System	Markov Chain	Hidden Markov Model
Controlled System	Markov Decision Process	Partially Observable Markov Decision Process

Table 2.1: Markov Model classification.

Typically, the Stochastic Power Models used in Dynamic Power Management fall into the Markov Decision Process category. What Markov Decision Processes (MDP) try to capture is the relation amongst sequences of actions in a system, and the state transitions that they cause.

Markov Models can also be further classified according to the cardinality of their time set and state space. Table 2.2 presents this kind of classification with more examples of stochastic processes.

	Discrete-time	Continuous-time
Discrete-states	Markov Process	Continuous-time Markov Process
Continuous-states	Harris chain	Wiener Process

Table 2.2: Markov Model Time Set and State Space cardinality classification.

As became apparent in Table 2.2, more types of stochastic processes exist. We will disregard the continuous-state processes, since the discrete-state ones are more relevant in the context of dynamic power management and the modelled state spaces are often-times described as a discrete set.

Learning Power Models.

"An agent is learning if it improves its performance on future tasks after making observations about the world" [Russel and Norvig, 2009]. This proposition is very relevant in to Dynamic Power Management, because there are some power management problems to which solutions are difficult to be programmed or even devised, due to the complexity of the systems at hand. In this way, the Policy can be conceived as an agent that learns a new Power Model from the data it gathers and actions it performs in run-time. This is why Machine Learning Policies tend to be both Power Model and System Model free, since they learn Power Models dynamically and they might not require any specific system information, in order to execute. They also tend to perform worse than policies that employ Heuristic models, though.

One particular type of learning process is *Reinforced Learning* (RL). In this case, the agent learns from a series of reinforcements: rewards or punishments. No direct consequence of the agent actions is observed, even though some feedback is provided in the form of hints, useful for the agent to reason on how it should operate.

It is often desirable to conceive Dynamic Power Management Policies that perform actions on a trial-and-error basis, learning from good and bad decisions. Hence, they can be designed as Reinforced Learning agents. One example of RL approach is found in the work of Shen et al [Shen et al., 2013].

A reinforcement learning model consists of three basic elements [Shen et al., 2013]: a state space that describes the environment, (or system status), an action space that defines the available control knobs and a cost function that evaluates the cost/benefit of different actions, given the state at which

the system is. How these three elements should be defined is determined by the available environment information, the nature of the system under control, as well as the user objectives and constraints. Therefore, it varies from problem to problem.

One common technique of Reinforced Learning is Q-Learning [Shen et al., 2013] (QL). Q-learning is designed to find stochastic policies, that follow the model of Markov Decision Processes (MDP). This technique is an iterative process with feedback from the previous iterations. At each step of interaction with the environment, the agent observes the environment and issues an action based on the system state. By performing the action, the system moves from one state to another. The new state gives the agent a *penalty* which indicates the value of the state transition. The agent keeps a value function $Q_p(s, a)$, (also called *Quality* of the state-action combination), for each state-action pair, which represents the expected long-term penalty if the system starts from state *s*, taking action *a*, and thereafter following policy *p*. Based on this value function, the agent decides which action should be taken, given the state the system is in, to achieve the minimum long-term penalties. As it is an iterative process, some initial numeral for the value function must be assumed, in order to start the algorithm.

To construct a Markov Decision Processes through Q-Learning, two questions need to be answered: (1) What are the states that compose the state-space? (2) How to formulate cost function, that depends both on the actions taken and states transited to, from the observed information?

System Models.

As shown in the particular cases of Heuristic and Stochastic models, Power Models often require information regarding the different power states in which systems can be. More precisely, it is often desirable to know how the power state transitions influence performance and the power consumption of systems. To that end, Power Models are often based on *System Models*.

System models are abstract constructs that describe how a system operates and prescribe functionality and interactions amongst different system components. They provide a basic framework of system behaviour, facilitating the conception of suitable power models.

An example of a System Model is the one of Service Requester and Service Provider, (SRSP in short). These systems are composed by four components: a *Power Manager* (PM), a *Service Provider*, (SP), a *Service Requester*, (SR), and a *Service Request Queue*, (SQ). The idea is such that:

- the Service Requester sends requests to the Service Provider;
- the requests are enqueued in the Service Request Queue;
- if the queue of the Service Provider is empty, the it is in idle mode;
- if the queue of the Service Provider is not empty, the it is not in idle mode;
- the PM is able to monitor service requests, and conclude the mode of the Service Provider;

Of course, typical real life systems have multiple requesters and multiple providers. This kind of models relax that fact, by considering that all the requests of all requesters come from a single source,

and the requests are also enqueued to a single point. Hence, one requester, one provider and one queue.

It is also common to consider more concrete systems as system models. One example is the work of Shen et al. [Shen et al., 2013] that considers for system model peripheral devices.

Adaptation.

Power models can be devised statically, before the execution of the policy, never changing or can be dynamically *adapted*, given the history that is maintained, in order to perfect the model, itself.

This is practical because systems workload changes over time, due to the number and type of applications running, users use and misuse of applications and other variable concerns that lead to chaotic and, sometimes, unpredictable power dissipation scenarios. In this way, adapted power models can be employed by policies, changing the criteria by which components are put to sleep or have their performance reduced.

Logically, every policy that employs machine learning techniques to devise its power model is an adaptable policy. Heuristic and Stochastic models can also be adapted in run-time, by any means other than Machine Learning.

One limitation of a dynamically generated power model is that it incurs in additional overheads. This is sometimes problematic, especially if the adaptation is computationally intensive or when there are tight performance constraints.

Synchronization.

The way the Policy communicates with the Observer and the Controller is a determining factor on how well the Dynamic Power Manager effectively helps to reduce the power consumption of a system's components. Therefore, it is relevant to classify a Policy regarding its communication *synchrony*, towards the other two DPM components, as synchronous or asynchronous. Typically, asynchronous policies perform better than their counterparts, since they do not incur in overheads as substantial as synchronous policies, by busily waiting for the observer's responses or the controller's actions to succeed. Therefore, they do not miss as many system events that can be relevant to the act of power management and operate in parallel with the Observer and Controller, enhancing performance.

Power Reduction Technique.

Policies can enforce the reduction of power consumption, according to different *technique* types. Either by selectively putting system's components to `sleep` or by *reducing the performance* of those same components. The notion of sleep state will depend of the system that is being managed. If the system under dynamic power management is an Unix based operating system, for instance, and the components to consider are processes, then a sleep state can be induced through a sleep() system call [Kerrisk, 2010]. Logically, the notion of "reducing the performance of" also varies from system to system. One common way of achieving lower power consumption through performance reductions is

through *Dynamic Voltage and Frequency Scaling*. DVFS [Weiser et al., 1994, Gerards and Kuper, 2013] allows the voltage of certain hardware components or the clock frequency of CPUs to be decreased, trading performance for energy. Current architectures provide mechanisms that allow direct access to system components, for DVFS purposes. Such is the case of Intel's SpeedStep [Rotem et al., 2004] and ARM's PowerNow! [Pow, 2000].

Policy Optimality.

Policy classification can be done with respect to *optimality*. Benini et al. [Benini et al., 1998] also point out that observation is indeed essential for devising good policies, i.e., it is strictly necessary to gather system data and adjust policy decisions in run-time. It is not sufficient to greedily put components to sleep as soon as they are idle. There are trade-offs involved that need to be considered. Namely (1) in case of multiple sleep states, the Dynamic Power Management System should choose one sleep state over the others and (2) since transitions to sleep-mode and back to active-mode also have a performance cost and inherent overhead, the DPM System should guarantee that the state transitions actually reduce power, compromising performance just up to an acceptable level. This leads to the problem of *Policy Optimization*, which is the one of choosing a Policy that minimizes power consumption, while under performance constraints, (or vice-versa), based on certain usage patterns. Such a policy is called an *Optimal Policy*.

The optimality of a given policy is always subject to the system model, in consideration, and the power model itself.

Relevant Dynamic Power Management Solutions.

In the work by Qiu et al. [Qiu and Pedram, 1999], the authors describe the problem of DPM as a continuous-time Markov Decision Process, applied to a SRSP system model. Other work, previous to this one, has some disadvantageous characteristics such as (1) considering time as a discrete dimension and (2) no notion of idleness in the modeled system components. This lack of accuracy would contribute to small energy gains because discrete models are limiting when managing real-time applications and because idle states are the ones at which power management actions should be enforced.

To overcome these disadvantages, Qiu et al. devised a Continuous Time Process and included the notion of idle and busy states of the Service Provider (SP). This is accomplished by adding a transfer state to the Service Request Queue (SQ), to represent the periods when the SP is busy, (since the SP accesses directly the SQ). The way request arrival and request service are modelled with Poisson distribution for the request arrival times, at the Service Requester (SR), and exponentially distributed request service times, at the SP. That lets the PM to be modelled as a event-driven component, thus reducing its decision making overhead, when put in practice.

The overall objective is to put the SP to sleep as soon as it enters the idle state. To do so, a Policy Iteration Algorithm is considered, in order to account for the performance constraints, inherent to the overheads of putting an SP to sleep (when they are idle), and wake the SP up (when they need to serve

an SR). On each iteration, a new policy is generated consisting on the cost of performing a sequence of actions, whose probability is weighted and summed to the delay cost of transiting from one system state to another (the actions could be, for instance to put providers to sleep or wake them up).

If the policy is optimal under the performance constraints imposed (an upper-bound to the cost function described), it is put in practice. Otherwise, a new iteration of the algorithm is performed, in order to adjust the sequence of actions that are to be made, and the respective delay costs state transitions.

In the work by Gerards et al. [Gerards and Kuper, 2013], the authors prove in a theoretical fashion in order to find an optimal schedule for a set of tasks it is necessary to consider both DPM and DVFS, instead of just maximizing idle periods length or minimizing clock frequencies independently.

They consider a system model of a number of periodic tasks, in which each of them is invoked the same number of times.

The authors conclude that it is best to either start each invocation as soon as possible or as late as possible, being this rationale used to find a globally optimal schedule that minimizes the energy consumption using DPM, for frame-based systems.

In the work by He et al. [He and Mueller, 2012], it is presented a simulated annealing (SA) based heuristic algorithm to minimize the energy consumption of hard real-time systems (real-time system where deadlines must be met) on cluster-based multi-core platforms. It is also proposed a technique that allows the power management algorithm to be executed in an online fashion, exploring the static and dynamic slack (times of idleness, or amount of time left until a new task is scheduled, during job execution).

The system model follows a classic real-time task model, since this solution is intended for multi-core systems. In this way, the system comprehends a task set, where each task corresponds to a pair of its worst case execution time and the deadline (equal to the period of the job the task is executing).

The main idea behind SA is to iteratively improve the solution by investigating the neighbour solutions, generated based on penalty and reward values obtained from the solution of the current iteration. If the number of iterations is sufficiently large, an optimal schedule of tasks can be found.

Shen et al. propose an approach [Shen et al., 2013] to dynamic power management using Reinforced Learning, specifically the *Q-Learning* algorithm. Even though QL can be applied as a model-free technique, the system under management is known before-hand, which allows for the enhancement of the QL algorithm. In this work, they propose a solution to the management of peripheral devices. As I/O devices they are, their workings are very similar to the Service-Requesters-Service-Providers model (SRSP), described in Section 2.1.2.2. To estimate the quality function $Q_p(s,a)$ of each state-action pair, it is considered the expected average power $Power(s,a)$ and request delay caused by the action a taken in state s, $q(s,a)$. The expected average power is computed as $Power(s,a) = \frac{(P_{A2B} \times T_{A2B} + P_{B2A} \times T_{B2A})}{2}$, where P_{A2B} and P_{B2A} are the power cost of changing from power mode A to power mode B and vice-versa, respectively. The request delay is computed as $q(s,a) = \frac{(q_{A2B} \times T_{A2B} + q_{B2A} \times T_{B2A})}{2}$, where q_{A2B} and q_{B2A} are the average request incoming rate during the power mode switching from power mode A to B (along the execution history of the system) and vice-versa, respectively. In this way, the policy chosen will consider states that minimize the delay cost at each state and expected average power wasted,

given the observations it has made, over the time the algorithm has been executing, while learning from its decisions and maximizing their quality. After a certain set-up time, the optimal policy can be found.

In the work of Wang et al. [Wang et al., 2011] the authors propose the use of Temporal Difference (TD) learning for Semi-Markov Decision Process (SMDP), as a power model-free technique, to solve the system-level DPM problem. Temporal Difference learning is a type of Reinforcement Learning. The system is modelled as a SRSP model.

A Semi-Markov Decision Process is similar to Continuous-Time Markov Process, with the exception that the decision maker can choose actions only when system changes state. Therefore, power management actions will be enforced only after the events that change the system's state. Temporal Difference Learning assume that the agent-environment interaction system evolves as a stationary SMDP, which is continuous in time but has a countable number of events. The periods at which those events occur are known as epochs.

The key idea is to separate time in decision epochs. At each decision epoch (corresponding to the SP being in a sleep state) actions are taken, depending on the state of the SR. At the next decision epoch, the action is evaluated in order to associate a value to the action taken previously. This will allow to chose from a set of power preserving actions, for each state of the SR, the one with the most beneficial value. Considering the number of requests from the SR and the total execution time to be fixed, the value function is equivalent to a combination of the average power consumption and per-request latency. The relative weight between average power and per-request latency can be changed, over epochs, to obtain an optimal trade-off curve between the average power and latency per-request.

In Table 2.3 the different algorithms previously presented are summarized according to the classification criteria established in the Section 2.1.2. The [-] symbol represents that a certain property is not applicable to a particular solution or that the authors did not specified anything regarding that property.

Paper	PowerModel	SystemModel	Policy			
			Optimality	Adaptation	Synchronization	Technique
Qiu et al.	MDP	SRSP	optimal	adaptable	asynchronous	sleep
Gerards et al.	-	Sporadic Tasks	optimal	-	-	DVFS
He et al.	Heuristic	Real-Time Tasks	optimal	adaptable	-	DVFS
Shen et al.	Q-Learning	Peripheral Devices	optimal	adaptable	asynchronous	sleep
Wang et al.	TD Learning	SRSP	optimal	adaptable	-	sleep

Table 2.3: Dynamic Power Management Schemes Classification.

2.2 Energy-Aware Scheduling

In the classical definition of scheduling, the goal of the scheduler is to determine which task, (thread or process), should be executed, according to some notion of priority. The idea is to optimize and take the most of CPU utilization.

Energy-aware scheduling is the problem of assigning tasks to one or more cores, so that performance and energy objectives are simultaneously met [Sheikh et al., 2012]. In this way, the goal of energy-aware

scheduling differs from the one of ”vanilla” scheduling, since it is intended to solve a multi-objective optimization problem, that comprehends both performance and energy.

Before studying different algorithms, two preliminary notions will be given, regarding the nature of multiprocessing systems and nomenclature.

1. From the perspective of scheduling, multiprocessor systems can be classified into (at least) two categories: (1) Heterogeneous processors are different in terms of architectural design. (2) Homogeneous processors are identical in design; hence the rate of execution of all tasks is typically the same on all processors. [Zhuravlev et al., 2012]

2. Each invocation to a task is called a *job*.

Different scheduling algorithms exist to manage a variety of system resources. We will focus on multiprocess systems, since our interest is to study the impact that performance and energetic constraints have on the execution of tasks. We start by studying the characteristics of some classical scheduling algorithms (Section 2.2.1) and, after that, we present energy-aware scheduling algorithms (Section 2.2.2), relevant to the work we pretend to develop.

2.2.1 Classical Scheduling Algorithms.

In the *First-Come-First-Served* (FCFS) scheduling algorithm [Yang et al., 2013], jobs are executed according to the order of their arrival time, to a waiting queue. The major disadvantage of this algorithm is the fact that large jobs greatly delay the execution of the next jobs to execute. This situation is called convoy effect.

The *Round Robin* scheduling [Tanenbaum, 2007] asserts to each job a time-slice where it can run. If a job cannot be completed in a time-slice it will return to the waiting queue and wait for the next time it is scheduled. Finding the proper value for the time-slices might be challenging to meet performance constraints. Even more if it is intended to achieve mutually performance and power optimization.

Earliest Deadline First [Jansen et al., 2003] is an dynamic scheduling algorithm where tasks are placed in a priority queue, such that whenever a scheduling event occurs the queue will be searched for the process closest to its deadline, to be scheduled to execution. Because the set of processes that will miss deadlines is largely unpredictable, it is often not a suitable solution to real-time systems.

2.2.2 Reference Energy-Aware Scheduling Algorithms.

In the work of Kamga et al. [Kamga et al., 2012], they propose a solution where they extend Xen default Virtual Machine scheduler – *Credit*. The goals are to (1) induce power reduction in the execution of several consolidated VMs while (2) respecting the agreed Service Level Agreement (SLA) – maintaining acceptable levels of performance.

Credit has two important parameters: *weight* and *cap*. Weight represents the priority of the VM and cap the CPU usage share, given in percentage. At least one virtual CPU (VCPU) is defined per VM. The scheduler transforms the weight into a credit allocation resource for each VCPU. As a VM runs, it consumes credit. Once the VM runs out of credit, it only runs when other VMs have finished executing.

Periodically, it is given more credit to each VM. The role of the extension consists in measuring the total VM CPU load, amongst all VMs, modifying each processor frequency through DFVS, every time the scheduling routine is executed.

The extension of the Credit scheduler is comprised of two modules: *monitoring module* and *cap control module*. At each tick, the monitoring module gathers the current CPU load for each VM and then computes the optimal frequency to which the CPU should be set to, according to the total VM load and the ratio between current and the maximum frequency. After that, the cap control module re-calculates new cap values for each VM, adjusting each VM CPU share to the fair percentage, taking into account the CPU load of each VM. Therefore, the objective of such re-calculations is to avoid performance degradation. In this way, it is possible to redistribute unused CPU cycles from one idle or less active VM to another, while minimizing CPU frequency to save energy, respecting the SLAs imposed.

Yan et al. propose an approach [Yang et al., 2013] where they introduce a job scheduling mechanism that takes the *variation of electricity price* into consideration as a means to make better decisions of the timing of scheduling jobs with diverse power profiles, since electricity price is dynamically changing within a day and High Performance Computing (HPC) jobs have distinct power consumption profiles.

Typically, user jobs are submitted to an HPC system through a batch scheduler, and then wait in a queue for the requested amount of system resources to become available. In particular, FCFS with backfilling is a commonly used scheduling policy in HPC, which might waste energy in an arbitrary way.

In this approach the scheduling system is composed by three components: a waiting queue, a scheduling window and a scheduling policy. The waiting queue is where jobs are stored in order to be processed by the HPC system. Rather than allocating jobs one by one from the front of the wait queue, the algorithm allocates a window of jobs. The selection of jobs into the window is based on certain user centric metrics, such as job fairness while the allocation of these jobs onto system resources is determined by certain system-centric metrics such as system utilization and energy consumption. By doing so, it is possible to balance different metrics, representing both user satisfaction and system performance.

The scheduling policy is intended to balance energy usage and scheduling performance and it is modelled following a 0-1 Knapsack based policy [T. H. Cormen and Leiserson, 2001]. Knowing that the overall objective is to reduce the accumulated power consumption during on-peak periods (high system load) and to increase the accumulated power consumption during the off-peak periods (low system load), the policy's goal is to minimize the value of the aggregated power consumption of nodes, during the on-peak period, and to maximize that same value, during the off-peak periods. Therefore the knapsack size is the number of available nodes, at the time of schedule and the objects that are to be put into the knapsack are jobs.

In the work of Datta et al. [Datta and Patel, 2013], the authors present two scheduling algorithms that address the utilization of homogeneous CPUs, operating at different frequencies, in order to lower the global power budget in a multiprocessor system.

The key idea explored in this work is that a task whose context is switched too often may not find valid data in its new core's cache, after being migrated to a new CPU. This task will have a tendency

to generate many cache misses. This overhead associated with cache coherence, and with context switching itself, can degrade the performance of a multi-core processor system. The scheduling is done by taking these facts into consideration.

By using *cache miss* and *context switch-CPU migration* indexes, the algorithms are able to exploit the increased performance associated with switching more computationally intensive tasks to higher frequency cores, without suffering from the performance losses associated with cache coherence and context switching overhead.

The algorithm assigns static and dynamic priorities to each task. For every initialization of a task, a static nice value is given to it, signifying its priority. Typically, tasks that are known to be CPU intensive require a lower nice – more priority. During the schedule stage, the algorithm moves computationally intensive tasks, that perform slower, to a higher frequency core or vice-versa, based on the number of context switches (or cache misses depending on which of the two algorithms is chosen) and the nice value.

2.3 Energy-related Certification and Analytics on the Cloud

The academic efforts to provide a means to certify web-pages in terms of its resource consumption are nearly non-existent, as far as we were able to find, even though it is a plausible idea to explore.

Here we start by analysing the current solutions that assign some sort of energetic rating to computational systems (Section 2.3.1). We then move to the cloud and big data systems domain (Sections 2.3.2 and
subsection:big-data-sys) in order to study the relevant work, that will give us insight on how to incorporate an energy-related certification sub-system into GreenBrowsing, following a cloud-based approach.

2.3.1 Energy-related Certification Computational Systems.

To our knowledge, there is no considerable work focusing on the energy-related certification of web pages. There is, however, some work that tries to rationalize and quantify the energy consumption of devices and software, for user visualization purposes.

Siebra et al. propose a scheme [de Siebra et al., 2011] to certify mobile devices, regarding their energetic performance. The idea of a green mobile certification is to use a set of test cases, which represent scripts of different mobile use patterns, to evaluate a mobile device. The evaluation is done based on mobile operations (voice call, Internet browsing, message services) and temporal delays between them. Each test case has an energy threshold that cannot be surpassed. If it is, then the mobile device under evaluation is not considered to be green.

Amsel et al. developed a tool – GreenTracker [Amsel and Tomlinson, 2010] – that aims at encouraging users to use software systems that are the most environmentally sustainable. They do this by collecting information about the computer's CPU and by comparing software systems in different classes of software (e.g. browsers are compared with other browsers), based on energy consumption.

For that effect, users are prompted to specify which classes of software they want to test. When all the systems in one class have been tested, Green Tracker creates a chart comparing the CPUs across all the software systems.

Camps et al. propose a solution [Camps, 2010] where a classification of web sites depending of their downloadable content is provided to users, making them aware of the web session costs. The classification is done statistically, by computing (i) the average size of objects embedded on pages, (ii) the rate flow and (iii) the distance from the web browser to the servers. The energy cost should be displayed to final user: this, from the authors perspective, will allow people to make smarter decisions on how to better manage their energy consumption in their web session.

From these three solutions, the most related to GreenBrowsing is, in fact, the solution presented by Camps et al. However, some disadvantageous characteristics make it less attractive than GreenBrowsing, in particular the fact that it only takes into account the downloadable content of web pages, disregarding important and predominant metrics such as *how heavy the page is* in terms of CPU, memory and I/O performance while rendering and executing JavaScript code. Moreover, all of the required statistical processing is done on the client side of the application, which might turn out to be a dominant overhead, leading to high resource usage and consequent energy consumption.

2.3.2 Classes of Big Data Analytics System.

There is a big variability in terms of Big Data systems that deal with energy data. In this section attention will be given to systems that gather home energy counters for auditing, analysis, and automation purposes.

Features. Singh et al. identify a number of features that can be used to classify a system, regarding its ability to aggregate data from multiple sources and to ubiquitously control data accesses and sharing (from any device and from anywhere) [Singh et al., 2013].

- Consolidation: To allow a single view into multiple data streams and cross-correlation between different time series, the system should automatically consolidate energy usage data from multiple sources.

- Durability: To allow analysis of usage history, a consumer's energy data should be always available, irrespective of its time of origin.

- Portability: To prevent lock-in to a single provider, data and computation should be portable to different cloud providers.

- Privacy: To preserve privacy, the system should allow a consumer to determine which other entities can access the data, and at what level of granularity, or employ mechanisms that preserve consumers privacy.

- Flexibility: The system should allow consumers a free choice of analytic algorithms.

- Integrity: The system should ensure that a consumer's energy data have not been tampered with by a third party.
- Scalability: The system should scale to large numbers of consumers and large quantities of time series data.
- Extensibility: It should be possible to add more data sources and analytic algorithms to the system.
- Performance: Data analysis times and access latencies should be minimized.
- Universal Access: Consumers should be able to get real-time access to their data on their Internet-enabled mobile devices.

Design Rationale. At the highest abstraction level, a system's architecture can be divided into the Data Store (D) components and the Application Runtime (AR) components, that access the data store, and perform the execution of analytic algorithms [Singh et al., 2013]. If we also consider that the system is comprised by two "endpoints" – one residing locally, at the client-side of the system and other residing remotely – three scenarios for the design of a system are possible:

- *Local-DataStore-Local-Runtime* (LDLR) - Both the Data Store and application Runtime are placed at the client end of the system. There is no remote end.
- *Local-DataStore-Remote-Runtime* (LDRR) - The Data Store is placed at the client side while the application Runtime is executed remotely.
- *Remote-DataStore-Remote-Runtime* (RDRR) - Both the Data Store and application Runtime are placed in the component of the system that operates remotely.

The main disadvantage of the LDLR design is that the application Runtime executes on the client side of the system, which can compromise system performance, due to the computationally intensiveness of the AR execution.

The LDRR design tries to solve the LDLR disadvantage by moving the application runtime to the component of the system that operates remotely. However, as it also happens in the case of the LDLR design, the *Consolidation* feature is harder to attain, since in order to integrate data from various sources into the AR functions, this would incur in greater complexity of the overall system management.

A RDRR design might releases the client-side of the application from the store and application runtime totally, providing a more lightweight approach to the client-end of the system than the LDLR and LDRR designs. However, by moving the Data Store to the remote end of the system, less control over personal data follows, because the granularity at which users can establish access permissions to their energetic data is greatly decreased. This introduces privacy concerns, since certain energy usage patterns might lead to the disclosure of personal habits the users do not intend to make public.

Business Rationale. This aspect reveals the purpose of the system, which can be classified as a Consumer-Centric system or an Utility-Centric one. The latter emphasizes on the usage of energy data by the system, in order to provide utility planning and operation services such as customer billing and

home energy waste visualization [Singh et al., 2013]. On the other hand, consumer-centric approaches emphasize consumer preferences regarding the way their data are handled [W. Liu and Pearson, 2011], by integrating their preferences in the decision-making of the services provided.

2.3.3 Relevant Energy-related Big Data Analytics Systems.

In the work of Lachut et al. [Lachut et al., 2012], they present the design of a system for comprehensive home energy measurement with the intent of automating the process of adapting energy demand to meet supply. They do this by measuring how the energy consumption is broken down by each appliance, on house, instead of measuring the overall energetic waste of all appliances or just at individual devices.

Instead of having one device measuring the energy consumed by each appliance, which might be considered intrusive, the authors state that only minimal collections of energy-related data need to be gathered, in order to measure the actual energy wasted at each appliance. Therefore only a small portion of the devices is installed at consumer houses. These devices will provide only the necessary metrics in order to statistically determine the energy consumption of each appliance, using a technique called Additive Factorial Hidden Markov Model.

The system is comprised by (i) Home Components that gather energy counters and send them to a (ii) Back End Server, which is responsible to apply the statistical methods necessary to measure the energy wasted, at different levels of granularity. Consumers can also visualize the energy wasted in their mobile devices through a Smart-phone Application, since the Back End Server provides a RESTfull API that these applications use.

In the work of Lee et al. [Lee et al., 2013], the authors propose an analytical tool that can assist in assessing, benchmarking, diagnosing, tracking, forecasting, simulating and optimizing the energy consumption in buildings. This tool is deployed in the cloud, in a Software-as-a-Service fashion, performing computationally intensive statistical operations on the data it gathers, and allowing for the visualization of energy-related data of users houses. The visualization is done at costumers devices through a dashboard application that summarizes the data outputted by the tool running in the cloud, alleviating any burden to the customer with regard to software maintenance, ongoing operation and support.

In the work of Singh et al. [Singh et al., 2013], it is presented a system that allows consumers to control the access to their energy usage data, from different devices on his/her house, and have it analysed on the cloud, using algorithms of their choice. The analysis of their energy-related data can be done by any third party application in a privacy preserving fashion. The system is separated in three main components: (i) Gateway, (ii) VHome and (iii) a variety of Applications. The (ii) Gateway is an home-resident and consumer-controlled component that collects home energy production and usage data that are uploaded, over a secure connection, to the cloud-based virtual home: the VHome component. It also provides an interface to allow the house owner to control devices in his/her house from Internet-connected devices. VHome is a virtualized execution environment hosted on a IaaS cloud provider. This (ii) VHome component is comprised by the Data Store and Application Runtime components described in Section 2.3.2. The (iii) Applications that can access the Data Store freely are the ones that belong to the

VHome application runtime. In order to allow other applications to access the Data Store, such as third party applications that can provide different analysis algorithms, privacy protection mechanisms (PPMs) are enforced. This PPMs pre-process data before it is transferred out of the VHome, by employing mechanisms like noise addition to the data transfered out of the VHome to these applications.

In the work of Balaji et al. [Balaji et al., 2013], the authors present a system called ZonePAC, for the energy measurement of modern houses with Variable Air Volume (VAV) type heating, ventilation, and air conditioning (HVAC) system and energy consumption feedback provision to the house occupants through a web application. The system basically makes use of existing sensors present on the deployed physical infrastructure of each building to communicate energy consumption counters from the sensors to a building management web service, called BuildingDepot [Y. Agarwal and Weng, 2012]. The communication is possible because a BACnet Connector (a user Computer) is adapted to the BACnet network. This network is formed of sensors and the BACnet Connector that communicate over a BACnet protocol. The BACnet Connector makes use of the RESTful BuildingDepot API to communicate energy counters gathered. BuildingDepot will act as a energy counter dissemination broker, since it informs the web applications subscribed on the available web services of new energy measurements. Finally, the users might check what are the energy consumption indexes of their in-house HVAC systems.

In the work of Oliner et al. [Oliner et al., 2013], the authors propose Carat, a system for diagnosing energetic anomalies on mobile devices. This system consist in a client application, running on a client device, to send intermittent, coarse-grained measurements to a server, which correlates higher expected energy use with client properties like the running applications, device model, and operating system. The analysis quantifies the error and confidence associated with a diagnosis, suggests actions the user could take to improve battery life, and projects the amount of improvement. The server is deployed in a cloud setting. There, the samples from client devices are sampled and analysed, aggregating the consumption of various mobile devices at the moment of analysis.

Table 2.4 presents the features that each system has. [*] means that a partial solution is given. [-] means that the authors give no information regarding that particular feature. Table 2.5 exhibits the classification for each system. To represent the fact that the authors gave no information regarding a specific classification property, the [-] symbol will be used.

System / **Feature**	Balaji et al.	Lachut et al.	Lee et al.	Oliner et al.	Singh et al
Consolidation	Yes	No	Yes	Yes	Yes
Durability	-	-	-	Yes	Yes
Portability	-	-	-	Yes	Yes
Privacy	-	Yes	Yes	-	Yes
Flexibility	No	No	No	No	Yes
Integrity	-	-	Yes	-	*
Scalability	-	-	Yes	Yes	Yes
Extensibility	-	-	Yes	-	Yes
Performance	-	Yes	Yes	Yes	Yes
Universal Access	Yes	Yes	Yes	Yes	Yes

Table 2.4: Big Data System Features.

System	Energy Data to Visualize	Design Rationale	Business Rationale
Balaji et al.	Home Energy Consumption	LDLR	Utility-Centric
Lachut et al.	Home Energy Consumption	RDRR	Utility-Centric
Lee et al.	Home Energy Consumption	RDRR	Utility-Centric
Oliner et al.	Mobile Device Energy Anomalies	RDRR	Utility-Centric
Singh et al.	Home Energy Consumption	RDRR	Consumer-Centric

Table 2.5: Big Data System Classification.

2.4 Analysis and Discussion

In this related work section, different energy and software related topics were covered, in order to understand how could a browser power management solution be devised. For each of the related work topics presented, some final considerations will be provided. These considerations will help to devise a suitable architecture for GreenBrowsing.

In Section 2.1 the trade-offs of Dynamic Power Management are explored, in order to understand the advantages of the different policies presented, as well as help perceiving the most advantageous situations where one could use those different policies. In particular, the concept of Dynamic Power Management – the exploitation of idle periods to optimize power consumption – resembles some of the intent of GreenBrowsing: to reduce the power wasted by idle tabs. With this in mind, it is important to consider that the more the policy adapts, the better the decision making is. However, adaptable solutions incur in bigger overheads and sometimes offer similar energy gains compared to more static solutions. Moreover, it is also important to have power management components that communicate asynchronously between themselves to avoid performance degradation.

In Section 2.2, scheduling was presented taking into account not only performance constraints but also energetic ones. The rationale of energy-aware scheduling is of great interest to the design of a multi-task architecture. It is important to separate concerns in a balanced fashion, in order to assign similar workloads to different tasks, in terms of the computational and resource access intensiveness of each job.

In Section 2.3, emphasis was given to the fact that it is desirable to move expensive and resource intensive computations to a remote system (e.g. in the cloud), when it comes to energy management. The data needed to do those computations can sometimes disclose private details of users. In this way, the information sent for remote processing should be as few, and as protected, as possible. In order to reduce the waiting time of resource processing on client applications, running on client devices, the server/cloud counterparts should execute as fast as possible to reduce the turnaround times of sending energy-related data to remote systems, processing it there and receiving it back. Therefore, the performance of remote data processing matters. Furthermore, remote systems should scale with the processing requests they receive. So scalability is also important to take into account.

Chapter 3

Architecture

There are two major subsystems that comprise the GreenBrowsing architecture: a Browser Extension that will act as a power manager, limiting browser access to resources (Section 3.1), and a Web Page Certification Back End, to be deployed as a prototypical big data analytics system (Section 3.2).

The Architecture of GreenBrowsing will be presented following the *Modules* and *Component and Connector* View Styles Garlan et al. [2010], loosely.

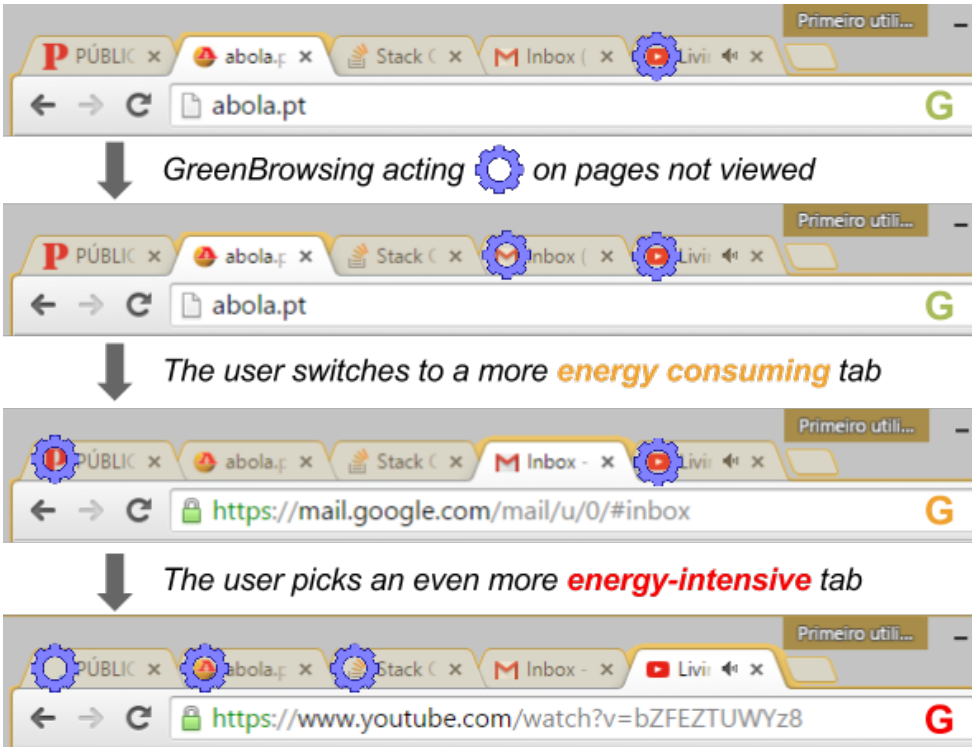


Figure 3.1: Symbolic depiction of GreenBrowsing actuating on idle tabs and energy-related web-page certification.

3.1 Extension Sub-System

The main roles of the Browser Extension are *to reduce the resource consumption of idle tabs*, and *to send to the Analytics back end resource-related data*, used to derive energy consumption data, in order to certify web pages in terms of their energy consumption while being accessed.

As in any discipline, trade-offs are often encountered. By enforcing a lower browser consumption in regards to certain types of resources, inefficiencies might arise if the reductions are too extreme, or if the slightest resource consumption variation greatly affects browser's expected performance.

To mitigate the impact of extension's operation on user experience significant requirements, such as user-perceived browser operational delays, different mechanisms will be presented and classified. This will provide a varied mechanism catalogue to evaluate (in Chapter 5), assessing what are the most prominent in reducing resource consumption rates while not impacting user experience.

Also, in order to properly separate concerns, a modular architecture will be adopted. This will allow the extension itself to be more easily extensible and easily portable across different browsers.

The next sections will start by introducing some of the terminology used along the document, getting then into the details of GreenBrowsing's extension, from a architectural standpoint. Finally, the algorithms and mechanisms used to perform resource usage limitation will be presented.

3.1.1 Terminology.

Throughout this document specific names will be used to refer to different browser entities. The definition of those entities is done as follows:

[Definition 1] Window. *Represents one graphical instance of a browser that contains one or more tabs opened.*

[Definition 2] Tab. *Most basic browser unit considered by GreenBrowsing to act upon. It is comprised of an unique id within the browser, and an unique index within its Window. It contains a web-page that is presented to users.*

[Definition 3] Focused Window. *The currently user-selected and visualized window. There may be many browser windows opened but* only one *focused window.*

[Definition 4] Active Tab. *A tab that is selected and whose web-page is being accessed by the user. There may be many tabs opened, within one window, but* only one *tab is active at a time.*

[Definition 5] Idle Tab. *A tab said to be is idle if it is not active.*

[Definition 6] Loading Tab. *If a tab is waiting its web-page contents to be loaded from a remote location (e.g. a web-server), or if it is waiting for its web page to be fully processed, it is said to be loading.*

[Definition 7] Complete Tab. *A tab is said to be complete if it is no longer loading.*

To further illustrate each of these entities as they appear on users' screens, a common arrangement can be observed in Figure 3.2.

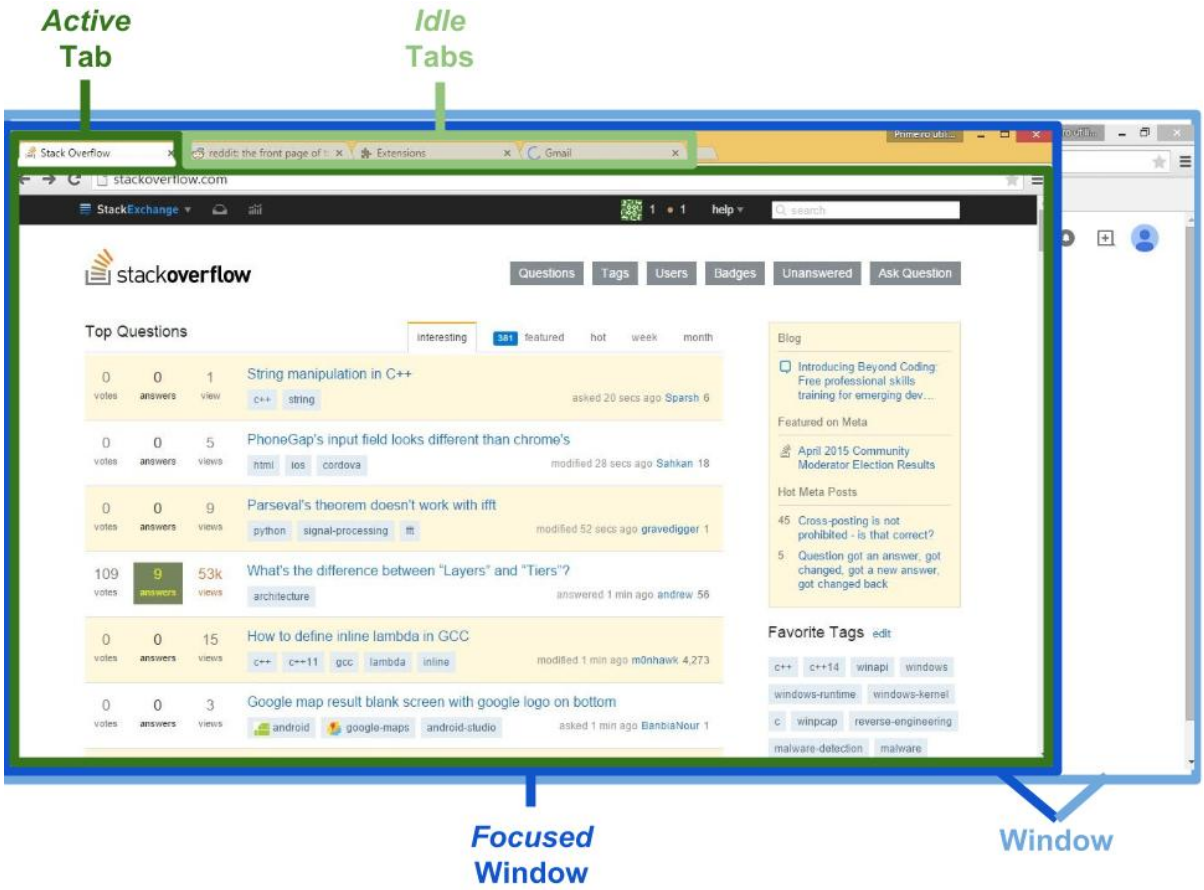


Figure 3.2: Disposition of Browser entities. All tabs are *complete*, with the exception of the right-most tab, which is loading.

The extension can *act* in different ways, once a resource limit is reached. Depending on the *action* taken, different ranges of values are expected for resource reduction, as well as for inherent user-perceived delays. In order to take those into account, the following definitions are introduced.

[Definition 8] Latency. *Latency corresponds to the time period that goes from the moment the active tab starts* loading *content to the moment it becomes* complete*. When the latency period ends, the active tab might not be the same from when the latency period started. For instance, if a tab starts loading content, a latency period starts. If the user switches to another tab (effectively changing the active tab) and if that tab is complete, the latency period ends.*

This notion of latency is useful to give an idea of how much time is wasted, by enforcing certain mechanisms, in comparison to others. The goal is, therefore, to cause the least possible overall latency, while trying to achieve minimum browser resource usage.

3.1.2 Modules and Run-Time Components Description

A Layered View of the Extension is presented in Figure 3.3. Each layer *uses*, exclusively, the layer(s) beneath it. Each of the modules is described as follows:

- **Observer-Controller-Adapter (OCA) -** It provides interfaces for gathering performance counters of each running tab and the process(es) it is associated with. It will also provide interfaces for issuing commands to tabs and the operating system itself, in order to reduce tab resource usage through the application of different mechanisms.

- **Certification FrontEnd -** Here is the code of the network communications that will need to be carried out with the Certification Back End, in order to send performance data of web pages to it.

- **Certification Renderer -** This module embodies the functionality needed to render energy-related rating of each web page, based on its certification, serving presentation purposes mostly.

- **Policy Enforcer -** Here the power reduction algorithm will be implemented. This module will need to use the OCA interface, to gather performance counters and to issue content adaptation and power reduction related commands.

- **Web Page Certifier -** This module will have code to fetch performance counters, through the Observer-Controller-Adapter. It will also interface with the Certification Front End to send the counters gathered to the Back End (for energy-related certification of web pages). Communications with the Certification Renderer are done to inform the user of each web page certification.

- **Profile Manager -** This module will have the code for the graphical interface the user might use to further tune GreenBrowsing to his/her preferences. Interfacing with the Policy Enforcer is done to communicate user preferences.

In terms of components, the execution of Policy Enforcer's code will be done in parallel with the control and content adaptation of tabs/pages, by two different tasks (comprising one or more threads, each). If they were to be executed sequentially, significant delays could occur in the policy's components execution.

There is also a dedicated set of threads running part of the Web Page Certifier code and part of the Front End code that will deal with the issuing of energy related data to the certification Back End. The other part of the code is run by another set of threads that will react (asynchronously) to the incoming certification rankings that come from the Back End certification system, avoiding waiting busily for those responses. Once received the certification stamps, these threads will also be responsible for running the Certification Renderer code, for the visualization of web page energy-related certification.

In order to avoid unnecessary round-trips, a cache of certification stamps will be maintained in the Extension.

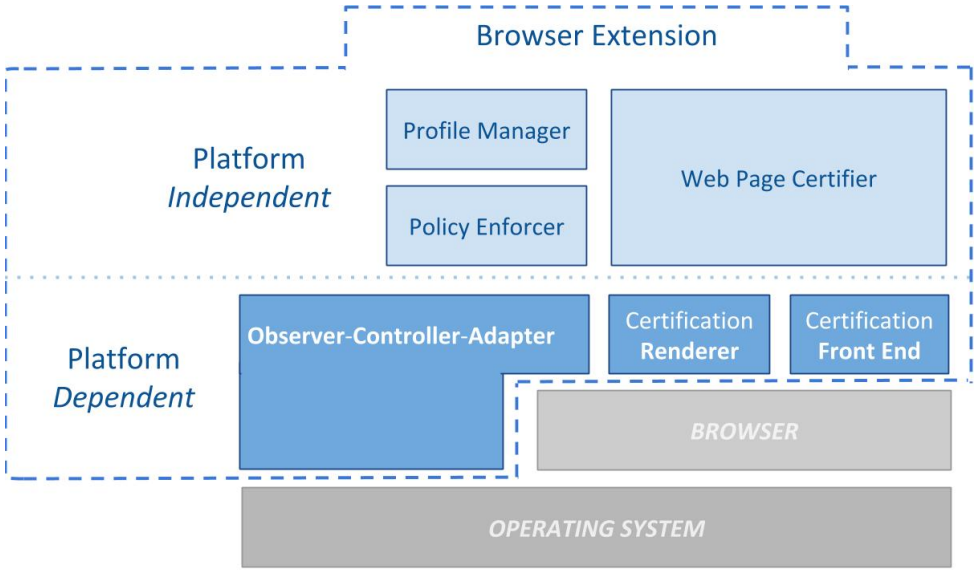


Figure 3.3: Layered View of The Browser Extension.

3.1.3 Tab Management Algorithm.

We consider that there is no best way to approach the problem of managing tabs for achieving power gains.

At a first glance, the complexity of the problem seems to require stochastic or machine learning-based techniques to suitably approach it, by progressively adjusting tab resource usage depending on user browsing habits. Specially due to the relative *unpredictability of user actions*. But even if those techniques were accurate, they are nonetheless computationally intensive in the great majority cases (as discussed in Section 2.1), introducing higher power consumption rates themselves, while trying to adjust their actions to users.

Having this into account, we conclude that we should approach this power management problem through simpler heuristics, that offer a smaller implementation overhead compared to stochastic or machine learning techniques. Specially because one of the requirements of this system is to cause the least possible user-perceived delays, while browsing the web.

Only two *assumptions* are made, regarding general browsing behaviour, serving as basis to the resource limiting mechanisms to be considered:

- **Last Time Usage.** Tabs that were accessed more recently are more likely to be accessed again and therefore will be less likely to be acted upon. In this way, the tab management policy will make use of a Least Recently Used list for tab energy management.

- **Active Tab Distance.** We also assume that tabs that are closer to the actual tab opened by the user are more likely to be accessed, therefore they will have lesser probability of being discarded or subject to resource constraints.

Since it is intended to manage power consumption of tabs, power consumption will be indirectly dealt with by lowering resource consumption. In particular, the memory each idle tab is using (in Mega-Bytes)

```
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
```

Table 3.1: Tab management algorithm overview.

and the CPU intensiveness (in terms of processor(s) percentual load) will be considered.

Considering a browser setting where N pages are opened, each one on a dedicated tab, with just 1 of them being visualized, there are $N-1$ idle tabs that can be acted upon for resource (and corresponding energetic) reduction purposes.

The tab management algorithm assumes there are at least two different processes mapped to two non-overlapping sets of tabs, this is, there is no tab handled by more than one process. The previous pseudocode summarizes the algorithm driving the behaviour adopted by the extension:

The first thing to notice is that *windows not focused*, this is, not selected by the user in case there is more than one window opened, have their tabs' halted, preventing them from executing (as described in Chapter 4).

For focused windows, in the other hand, the basic idea is to allow an active tab to consume resources, while penalizing all the other idle tabs, according the assumptions of *last-time-usage* and *active-tab-distance* described previously.

Every time a tab becomes active, the algorithm is run, updating the facilities responsible to keep record of what limit restrictions are imposed on each tab.

The following formula is used to set the maximum resource usage (for any resource type), 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 $p >= 1$, $i >= 1$, $a >= 0$:

$$resource\_usage\_factor(i, p, a) = \frac{1}{p \times i^a} \tag{3.1}$$

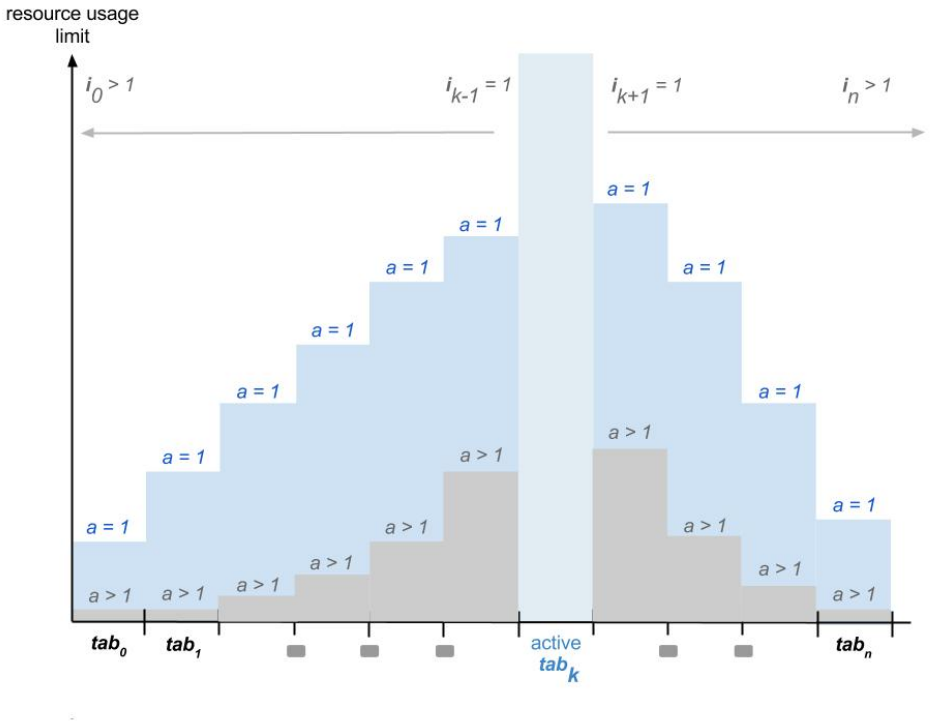


Figure 3.4: Effects of tab management algorithm on tab resource consumption.

The value computed with the previous formula will be then multiplied by the maximum possible resource type value, that a certain *idle tab* can consume, under the influence of any given resource consumption mechanism. The intended effect on focused windows' tabs resource consumption is depicted in Figure 3.4.

It is possible to have two tabs at the same distance i from the active tab and still experience different resource usage limits, for the same value of aggressiveness a, since one of them could have been activated more recently (holding a smaller value for p).

One final remark is that some idle tabs may share the same process with the active tab. If this happens, those idle tabs will not be acted upon, since the resource consumption of their process would also constrain the active tab's resource usage, (and possibly degrade user experience).

3.1.4 Mechanisms for Resource Reduction.

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

In this way, terminology presented at Section 3.1.1 is extended with designations a process might have, depending on its browser-related role:

[Definition 9] Renderer Process. *It is the type of process responsible for processing web-pages contents of one or more tabs. One tab is associated with only one renderer, though.*

[Definition 10] Browser Kernel Process. *It is the main process that orchestrates all browser activity. Render processes access resources through it.*

[Definition 11] GPU Process. *Process responsible for displaying GPU-accelerated content. In particular, it is useful for relieving the power consumption inherent to CPU-intensive task by migrating them to devices' GPUs, since GPU often consume less energy over time.*

These definitions are in accordance with the terminology found in the work by [Wiltzius, Tom et al., 2014]. In order to reduce resource consumption, different mechanisms can be applied and, depending on the type-of-resource/metric considered, the maximum allowed value for idle tabs will vary from tab-to-tab. *Four* resource metrics are considered in GreenBrowsing, to be applied on a per-process basis:

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:

$$round(resource\_usage\_factor(i, p, a) \times x) \tag{3.2}$$

   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.

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:

$$round(resource\_usage\_factor(i, p, a) \times 100) \tag{3.3}$$

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:

$$round(resource\_usage\_factor(i, p, a) \times max\_memory\_committed) \tag{3.4}$$

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, which is might not be considered a resource per-se, but its control might induce the computational resource reductions beyond CPU and memory, since tab processes may, for instance, be required to do I/O or networking activities.

   The adjustment formula for allowed process execution time is compute as:

$$round(resource\_usage\_factor(i, p, a) \times average\_tab\_activation\_time) \tag{3.5}$$

Once a certain limit is hit – i.e. the maximum value for a tab to consume was reached or surpassed – the effects on the tab depend on the type of mechanism employed. The effects expected once limits are violated are described as follows:

1. If **prio** is active, there is no concrete action taken, because changing process execution priorities is not, in itself, a resource limiting mechanism. The expected outcome would be, however, for a tab's process to execute less often relative to other processes (browser or any other application's related). But, indeed, the arbitration of when a tab's process should be executed is delegated to the Operating System's scheduler, entirely.

2. If **cpu** is active, once a tab's process processor 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. For **mem** there are two versions of this mechanism, with two different possible effect outcomes, once a memory limit is reached by a process:

   - Soft version: the process is either halted, and put to a sleep state, returning to execute once its tab becomes active, or
   - Hard version: the process is terminated, releasing all the resources allocated until then.

4. 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.

By combining the four resource adjustment metrics with the effects on resource usage limit violation, described previously, a total of *six* mechanisms are singled out. Table 3.2 summarizes these mechanisms in terms of the metric that is directly adjusted by the mechanism, the maximum value for resource limits and action taken on limit violation.

Model	Metric	Maximum Resource Value	Action on limit violation
prio	cpu usage	Normal process priority	-
cpu	cpu usage	100 % usage	postpone execution
mem soft	memory usage	maximum memory committed	halt execution
mem hard	memory usage	maximum memory committed	terminate process
time soft	execution time	average tab activation time	halt execution
time hard	execution time	average tab activation time	terminate process

Note: to *postpone* execution, in this context, means to stop executing and re-execute again once the OS scheduler intends, i.e. to yield the process. To *halt* the execution means to re-execute just once a process' tab becomes active.

Table 3.2: Mechanisms summarized classification.

3.2 Analytics & Certification Back End Sub-System

The Certification Back End Sub-System has the objective of *providing a clear and meaningful notion of how much energy web pages consume*. It also certifies domains, as a way to alert users of web-sites that generate resource hungry, power consuming web pages.

As a way to reduce the computational intensiveness required by the extension, the energetic certification of web pages is moved to a separate subsystem, that is intended to be deployed remotely (for instance, in a Cloud Computing setting).

In this section, the Architecture of GreenBrowsing's Back End will be presented and explained. After that, the *Certification Modelling* stage of the web-page URL+domain certification process will be described, as well as the *Certification Algorithm*. Finally, a description of resource usage gathering will be given, detailing what problems might arise if not done properly.

3.2.1 Components of the Certification Sub-System.

The Certification Back End will have, at least, the following components (as depicted in Figures 3.5 and 3.6):

- A **Certification Server**, comprised of *Network Communication tasks* that receives energy-related web page certification requests and forwards these requests to tasks specialized in the certification of pages themselves (to avoid service bottlenecks and enhancing the scalability of the system regarding the treatment of requests); those are *Analytics Certifier tasks*, that do the work of certifying a given page, according to a specific certification model.
- A **Certification Modeller**, comprised of *Certification Modeller tasks* that adjusts the certification model, having into account all the resource data sent from the extension subsystem. For performance purposes, this design emphasizes the usage of specified *Worker Tasks* to whom parts of the analytical calculations are mapped to. The results of processing data at workers are assembled back at the Modeller Task, as soon as they are ready.
- A **Data Store** that stores the models used in the certification of pages and tuples with information relative to the performance counters of each page;

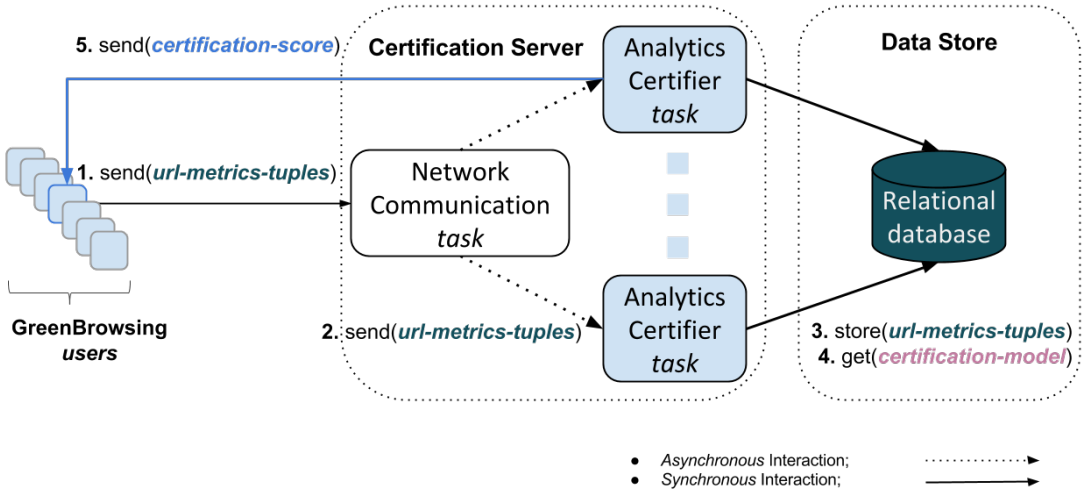


Figure 3.5: Certification requests sent from GreenBrowsing users to the Certification Server.

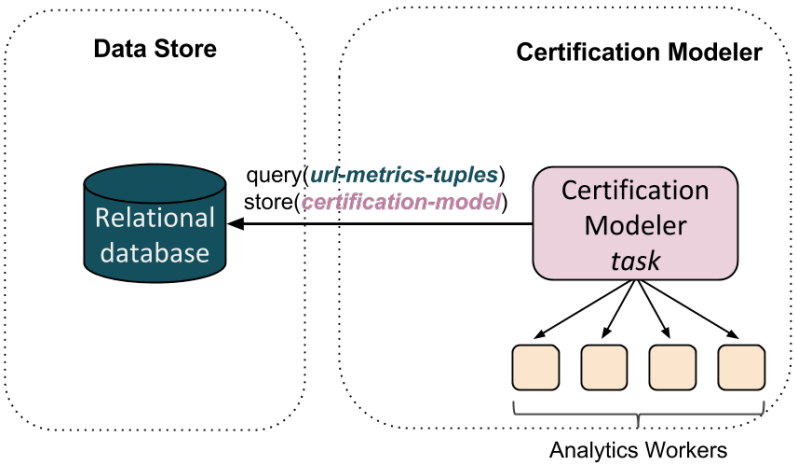


Figure 3.6: The Communicating tasks of the Back End.

3.2.2 Performance Counters for Energy-related Certification.

The power consumption induced by web pages will be indirectly determined by some of the performance counters gathered on the Browser Extension. For each page, the metrics considered will be:

1. CPU usage (in terms of completed clock cycles);
2. Private (main-)memory usage of processes (in Mega-Bytes);
3. 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 (Rodrigues [2013], Bircher and John [2012], Park et al. [2009]).

The certification will be done at the level of the *web page* and *domain*, but could easily be extended to individual subdomains, subtrees of each domain hierarchy, for instance. Therefore, the information sent from the Extension to the Certification Back End will be a 5-tuple $<$*id*, *type*, *CPU-usage*, *memory-usage*,

network-bandwidth-usage$>$, where the *type* entry indicates if the performance counters refer to an URL or domain and the *id* refers to its textual representation. For each page, two tuples will be sent to the Back End, periodically, typically in one minute intervals, approximately.

These metrics will be sent to the Back End, after a page is rendered (accounting for the resource consumption of JavaScript just-in-time compilation and HTML+CSS processing) and before the tab of a certain page is disposed by the power management threads (accounting for the resource usage due to user activity).

3.2.3 Requirements for Energy-related Certification.

Along with placing the certification process on a remote set of servers, other requirements regarding the certification process, itself, should hold. In principle, the act of certifying something implicitly considers the existence of a set of well-define ranks or certification categories, which in their totality are all-inclusive to any web-page, i.e. given a certain web-page it is always possible to associate a energy-related classification to its URL/domain. This might not be trivial, since many different resource usage patterns are expected to be observed while processing web-pages, due to the variability of web technologies and richness of web content. This, also because not all resource consumption behaviour inherent to web-page processing is known.

While devising a certification scheme, one should also consider that the entities to certify change over time. Web-pages are no different. What might be considered resource intensive in the present, might be considered acceptable in the future (or, most likely, the other way around).

So, in essence, the requirements expected for an appropriate certification scheme, in the context presented, are:

1. Group resource consumption from various sources to ensure all-inclusiveness of certification categories;
2. Predict unobserved resource consumption ranges to further ensure completeness/inclusiveness of certification categories;
3. Dynamically adjust the certification scheme to the changes in web-page properties, that induce varied resource consumption patterns over time;

3.2.4 Devising Certification Categories.

The requirements were set, but there are questions left to answer, still: (i) How to devise certification all-inclusive categories from multiple sources? (ii) Given two categories, which one dominates, or better yet, which one represents the *greener*, associated with less resource consumption patterns?

To answer the first problem the Certification Modeller algorithm employs a clustering method, based on the work of [Dempster et al., 1977] – named *Expectation-Maximization*(EM) – in order to find no less than 8 categories of certification (clusters). This is done in a 3-dimensional (multivariate) random variable space that comprehends one dimension for the CPU usage, one for the memory usage and another

Input: A set $O = \{O_1, O_2, \ldots, O_n\}$ – resource consumption values
Output: A set $C = \{C_1, C_2, \ldots, C_k\}$ – cluster centers of mass
$C \leftarrow \{C_1, C_2, \ldots, C_k\}$, $C_i \in \mathbb{R}^3 \ \forall i \in \{1, k\}$
$P \leftarrow EM(O, k)$, $P = \{P_1, P_2, \ldots P_k\}$, $P_i = \langle \mu_i, \sigma_i^2 \rangle \ \forall i \in \{1, k\}$
foreach $i \in \{1, k\}$ **do**
  $S_i \leftarrow MGMM_{sample}(P_i)$
  $C_i \leftarrow CM(S_i)$
end
return C

Table 3.3: Algorithm for modelling Certification Categories.

for network usage. 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 ($MGMM$) are used to fit the data and to iteratively train the parameters for 8 random variable's sub-populations, each one corresponding to a cluster. The parameters in question are:

- a 3-dimensional vector comprising the means of each random variable and
- a 3×3 covariance matrix;

After having trained a group of MGMM clusters, a random selection of trained cluster observations is selected from each cluster. The center of mass (CM), or centroid, of each sample is computed, afterwards. The resulting center of mass vector obtained this way, is representative of the category, identifying it unequivocally, and will be used to certify web-page URL or domains while running the certification algorithm. In order to qualify a certain cluster, the vectorial norm of the hypothetical vector space origin to the center of mass of that cluster will be considered. The greater the norm, the more resource intensive pages with that norm's certification category will be considered to be. This is done once, per trained model.

The method previously described is summarized in Algorithm 2. The input consists of a set comprised of n resource consumption values gathered from many users' devices, from which k certification categories will be devised and returned.

Since the EM algorithm is an efficient iterative procedure to compute the Maximum Likelihood estimate, in the presence of missing or hidden data, and since data is intended to be grouped from as many user devices as possible (while running GreenBrowsing Browser Extension), requirements 1. and 2. are met. The 3rd can be ensured by re-computing samples' CM periodically, following the approach described.

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 $\langle s, k \rangle$

Table 3.4: Certification Algorithm used to score web-page URL and domains.

3.2.5 The Certification Algorithm.

In order to certify a pages URL and domain, tasks running at the Certification Server fetch the clusters' centers of mass, of the last trained Certification Model, from the Data Store. After that, the certification algorithm will take as input those centers of mass and the last observed measurements for any web-page URL/domain. These measurements are included in the certification request issued by GreenBrowsing's Browser Extension. Measurements are gathered by the Extension, as well.

The algorithm to certify a URL/domain's web-page with respect to its consumption consists in comparing the Euclidean distance (d) that goes from each observed resource measurement to the center of mass of each cluster. If two or more clusters' centers of mass are at the same distance from an observation, the one with the greater norm is associated with the observation. In the end, the cluster/category that is associated with more observations, is the final certification category assigned to the URL/domain.

The certification methodology is described more succinctly in Algorithm 3. The input consists of a set of n resource consumption values gathered from a single user device, and a set of k certification categories, previously computed, according to Algorithm 2.

3.2.6 Resource Usage Gathering.

The way resource usage is gathered by the browser extension on user devices, influences the certification score and, most importantly, the certification parameters obtained during the modelling phase.

One of the parameters to determine, for each cluster, is a covariance matrix. One detail of Multivariate Gaussian Mixtures is that each of the mixture's clusters covariance matrices must be non-singular, otherwise their determinant would be null, resulting in a division by zero in the expression of the normal distribution's probability density function, that describes a certain cluster. To avoid such situations, the

resource values obtained on the extension are non-monotonic. This is, any given observation must have a value different from the one previously recorded. In this way, data sets of observations offer more value variety, potentially inducing bigger variances and avoiding obtaining singular covariance matrix.

In order to preserve user privacy, the URL and domain identifiers sent to the Back End will be partially anonimyzed by hashing them independently. This will also enable fast indexing/search, after being stored remotely, while providing significant privacy, since it requires extensive brute force to extract the URL/domain, given an hash of it.

Chapter 4

Implementation

In this chapter, the details regarding the realization of the Extension and the Back End are described, emphasising the difficulties encountered and respective solutions. Exposition of alternative approaches, that could have been chosen over the devised implementations, will also be provided, where appropriate.

4.1 Browser Extension

In order to and manage tab processes, operating system abstractions need to be manipulated. After all, a process only exists in the execution context of the operating system it resides in. Since Chrome has very limited support for process management, namely of its tabs, the Extension needed to be divided in two main entities:

- **The Browser Extension** itself, comprised of JavaScript callbacks and code rather event-oriented, whose execution and handling is delegated totally to the Browser, by running from within the Browser itself as a Google Chrome Extension.
- A **Background Process (BP)** running natively as a Windows application. Through it, browser processes can be directly managed by communicating, beforehand, with the extension.

These two components will be scrutinized, with certain detail, in the following sections.

4.1.1 Browser Extension & Background Process.

The Extension communicates with the Background Process issuing mechanism-related commands and in order to allow the latter to keep track of certain browser state, relevant to the Tab Management Algorithm described at Section 3.1. The browser state-related information passed this way is composed of general tab information such as tab identifiers, tab indexes within their windows and corresponding process ids. All communications are handled asynchronously by the Background Process each time an event is raised by the browser, following a certain tab state update. For instance when a tab is created, or when a tab is activated. A Communication Handling Thread is kept at the BP, blocking itself while waiting

for new messages coming sent by the Extension. Once a new message arrives from the browser, on behalf of the Extension, the Communication Thread unblocks, reading and parsing the message and passing it to the main thread, where the respective command is processed.

Consider the case of a tab being *created* and becoming *activated*, in the process of creation. In some cases, the first message to be received at both the Extension and, after that, at the BP is the one representing the activation command. This is problematic because there is, still, no idea of what is the process associated with that tab, since from the BP perspective, a creation command was not received for that tab, yet. This is just one example of the event-oriented and unpredictable behaviour of the Extension. If left unchecked, these situations could cause failures and erratic behaviour on the BP. To account for this, fault-tolerance enhancements were implemented on the Background Process' code. Those enhancements are summarized in the flowchart of Figure 4.1

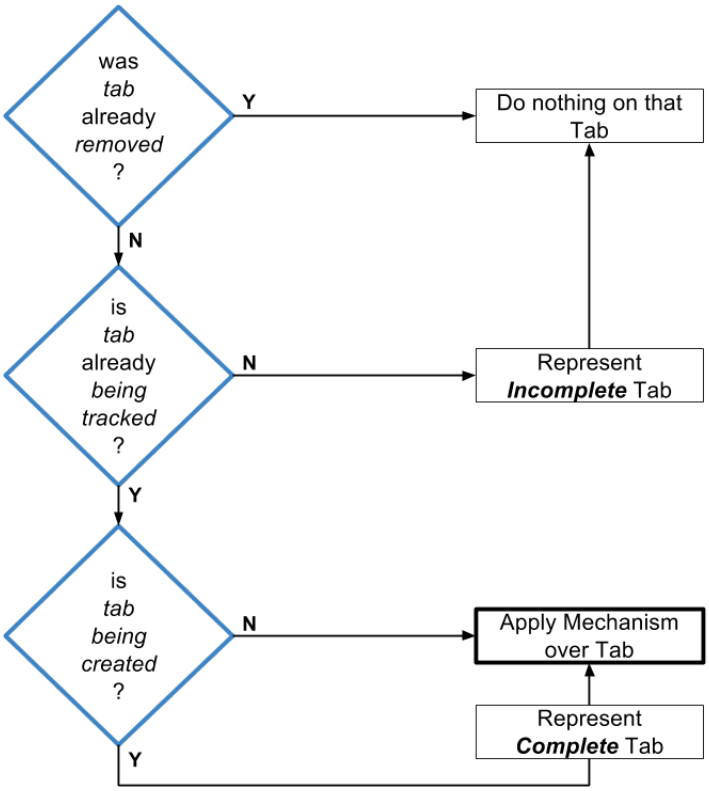


Figure 4.1: Tab State Consistency Flowchart.

Basically, mechanisms can only be employed on a certain tab if the state of that tab is known and it is totally kept at the BP. Knowing that tab removal events can also be received (at the BP) before any other event, if a tab was already removed, it makes no sense to try to act over it. The idea is to first check if a certain tab has been already removed. If not, if it has state missing and the event received (at the BP) states that a new Tab was created in the browser, all state relative to that tab is also received with the event, being safe to apply mechanisms to that tab.

4.1.2 Operating System Facilities.

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 3.1, for the purpose of resource and energy consumption reduction. Since associating more than one Job object to a single process is only possible in newer versions of Windows, (this was available only from Windows 8 on), and because associating more than one Job to a single process results might have non-intended outcomes, such as tampering with limits established by other Jobs associated to a single process, it was decided to retrieve the Jobs associated by Chrome to tab processes and use them for the purposes intended with this work.

The Sandbox model used in Chrome prescribes the association of a single process to a single Job. But, it is not clear where those Jobs are kept, i.e. in which of the Chrome processes' memory space the Jobs that encapsulate tab processes are. In order to discover what processes kept the Jobs needed to manage tabs' processes, it was necessary to enumerate all of Chrome's Windows kernel objects. This, in turn, was made possible by utilizing a set of undocumented functions "hidden" on the Windows API, which revealed rather challenging due to the lack of information regarding such functions. Finally, after overcoming those difficulties, it was possible to verify that tab processes' Jobs were, in fact, solely kept at Chrome's Kernel Process (this process is the one as defined in Section 3.1).

Knowing from which process to draw tab processes' Job Objects from, it became possible to act upon tab's processes in terms of restricting process resource consumption. The sequence of actions taken, each time a new tab is created, are described as follows:

1. Check if a certain process id is already being tracked – this happens if a certain group of tabs is rendered by more than one process. If it is, the Job Object is already known and nothing else needs to be done. If it is not known, the Job must be searched for.

2. Enumerate all of Chrome's Kernel Process Windows Objects;

   (a) Select those that are Job Objects;

   (b) Keep all Jobs found for the next step;

3. Check, for each Job found, if it is associated with the process responsible for rendering the tab that was just created. This is done by calling a specific documented function of Windows API named `IsProcessInJob`;

4. When the Job is found, the search is over and the process is ready to be acted upon.

4.1.3 Soft Mechanisms.

Each mechanism is implemented by exploiting the capabilities of Job Objects. For instance, it is possible to chance process priorities or adjust maximum CPU rates for any given tab process belonging

to a single Job Object. This is accomplished in the cases of *prio* and *cpu* mechanisms (described in Section 3.1).

When it comes to *soft* mechanisms though, more needs to be done. Recalling the effects that *mem soft* and *time soft* have on processes, once their adjusted limits are reached, processes have their execution halted. In practice, one can halt a process, by halting all of its threads. Since the functions involved in doing so are well documented, this is something accomplished in a straight-forward way. The most difficult part of *soft* mechanisms consists in knowing when Job limits, associated with tab processes, have been violated, because contrary to *cpu*, for instance, actions taken on limit violation are delegated to the user-program instead of the Operating System. As consequence, there are at least *three* ways of knowing whether Job limits were violated:

1. Associating a single I/O Completion Port (IOCP) to a single Job object in order to know when a certain limit was reached (Windows prohibits associations other than on a one-to-one basis, between Completion Ports and Jobs). IOCPs are basically a Windows abstraction for asynchronous message queues. Some processes post messages to IOCPs while others might dequeue messages at any moment, blocking until messages are put onto the IOCP queue. This method will not work in the case of Jobs created by Chrome, since every time an association between a GreenBrowsing owned IOCP and a Chrome Job is tried, it fails. This can only mean Chrome already associates Completion Ports to its Jobs.

2. Retrieving IOCP objects (knowing Chrome associates IOCPs to its Jobs), the same way Jobs are retrieved (through Windows Kernel objects enumeration), and have a dedicated thread attempting to dequeue messages from a certain Job's Completion Port. This is however quite destructive since it tampers directly with Chrome functioning. Chrome actually ”freezes” on a successful message dequeue from any Completion Port retrieved. Therefore, implementing such a solution is out of question.

3. Periodically checking whether limits were surpassed, by directly querying each of the tabs' Jobs. This is not a good solution, since the thread needs to be put to sleep in-between limit checks, not to waste to much CPU. It, unfortunately, is the only solution found for the problem of knowing when the limits of Job objects, adjusted through *soft* mechanisms, were violated.

4.1.4 Hard Mechanisms.

As mentioned before, certain tab state needs to be kept on the Background Process to effectively apply the resource reduction mechanisms intended. The typical course of action includes user manipulation of tabs and propagation of those manipulations from the Browser to the Background Process, e.g. tab creations, tab removals, etc. When employing *hard* mechanisms, tabs' processes are terminated once they reach their limits. The problem is that the Browser does not issue events to the Extension once a certain tab's process is terminated. If these situations are not accounted for, strange behaviour might follow the employment of the Tab Management Algorithm. Examples are the enforcement resource re-

duction mechanisms on the active tab or the erroneous computation of limits for idle tabs, as depicted in Figure 4.2.

Therefore tab processes need to be checked for their current status. In practise this is achieved by checking if a certain process' Windows objects are signaled, i.e. waiting on it for some reason, just like a barrier, because once a process is terminated its handles to become signaled. In code terms, this means calling `WaitForSingleObject` and checking if the return code was `WAIT_OBJECT_0`.

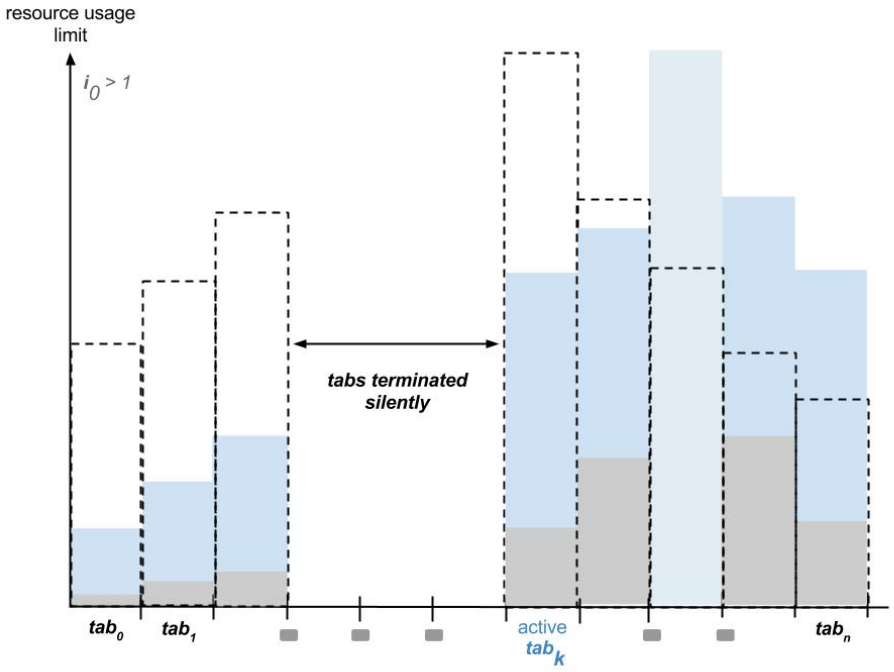


Figure 4.2: Faulty effects due to silent Process Termination.

4.2 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 Netty-socketio framework, to serve incoming certification requests. This framework is an implementation of the WebSocket protocol and allows to serve requests efficiently and asynchronously. Figure 4.3 shows how this is accomplished more in detail.

The basic idea is to have a process – Certification Server – interacting with Netty-socketio's runtime. The latter assigns Certification Threads, for each user resource consumption record received, (possibly re-utilizing threads from a pool). The records are passed to the Certification Threads on instantiation. The resource consumption data is then sent to the Data Store. If the resource consumption record was received together with a certification request, the Certification Thread fetches the Certification Model from the Data Store (comprising the information needed to certify URL and domains). The Certification Thread is now able to certify a web-page's URL and domain, using the resource consumption record and the model parameters received previously.

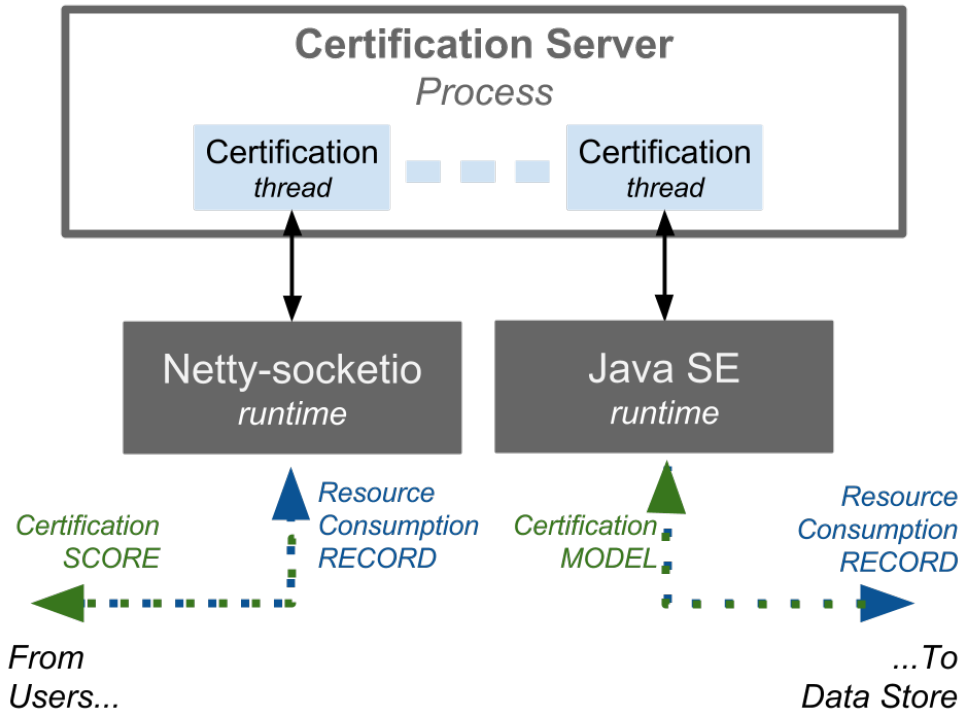


Figure 4.3: Certification Server Process and Threads.

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. The SQL tables that comprise the database schema are summarized in Tables 4.1 and 4.2. The first represents the structure of resource consumption records, gathered at users' devices. These are the records that will be fetched by the Certification Modeller to devise the certification categories. The latter represents the centroids used in the process of certifying URLs and domains.

Resource Consumption Table					
RowName	**Hash**	Type	CPU	Mem	Net
SQL Data Type	String	String	Array	Array	Array
Description	Record identifier	URL or domain	CPU usage	Memory usage	Bandwidth usage

Table 4.1: Resource Consumption Table details. Row names in **bold** represent primary keys.

Model Parameters Table				
RowName	**Type**	CPU_CM	Mem_CM	Net_CM
SQL Data Type	String	bigint	bigint	bigint
Description	URL or domain	CPU dimension centroid	Memory dimension centroid	Bandwidth dimension centroid

Table 4.2: Model Parameters details. Row names in **bold** represent primary keys.

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. The previous description can be visualized in Figure 4.4.

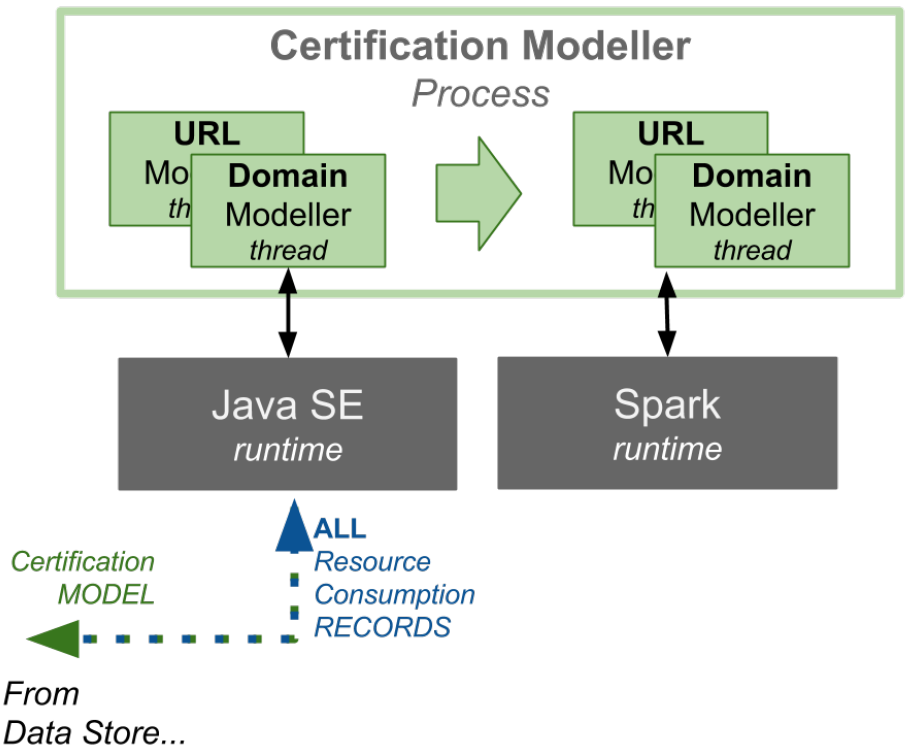


Figure 4.4: Certification Modeller Process and Modeller Threads.

It is possible to see that *all* resource consumption records are fed to the URL and Domain Modeller threads. After successfully receiving the records from the data store, each modeller thread unmarshals the JSON-formatted records and writes them to disk, as a monolithic file, structured as a set of lines, each of which contain a value for CPU usage, other for memory usage and another for bandwidth usage. Each line represents a record, in essence. Once this is done, the modeller threads start an instance of Spark, for each one of them. The expectation-maximization functions are now able to be run, followed by the sampling function. Finally, the modellers return, to the Data Store, the Centers of Mass for each of the sixteen URL + Domain categories/clusters. The certification cycle repeats again, starting from the point where, the most recent, resource consumption records are fetched from the Data Store.

4.3 Final Implementation Considerations

Energy efficiency is usually taken into account to drive resource and task scheduling in cloud environments Sharifi et al. [2014]. Decisions to rule resource management and constraints could be expressed resorting to declarative policies easily expressed in XML Veiga and Ferreira [2004] instead of setup in the code. Regarding mobile devices, more prevalent today, monitoring and management can leverage previous efforts on harvesting computing power from mobile devices de Oliveira e Silva et al. [2008]. If addressing the JavaScript execution environment for Chrome, for each task, available resources could be further monitored and managed according to previous work in Java environments, taking application progress into account Simão and Veiga [2012]. The Back End processing can be made more efficient when manipulating cloud storage with divergence bounding guarantees Esteves et al. [2012], and with incremental processing on top of such cloud storages Esteves et al. [2013, 2014].

Chapter 5

Evaluation

Throughout this document, special emphasis has been made on how energy consumption was proportional to computational resource consumption. Hence, the evaluation in will be done by measuring resource consumption under different circumstances. This also has to do with being difficult to separate the actual energy wasted by the browser, with and without our extension, from noisy energy patterns caused by other applications and system activity.

Furthermore, when it comes to evaluating the mechanisms presented, even though some of them act directly upon certain resource metrics, they almost always have collateral impact on other metrics not targeted. These situations are also taken into account, in the evaluation phase.

In this chapter the evaluation of GreenBrowsing's Extension will be made, in regards to: (1) Extension-induced resource consumption variations, comprising gains and losses relative to not using the extension; (2) Latency caused by different resource reduction mechanisms; (3) Correlation between Latency and Resource Usage patterns. Latency is defined as in Section 3.1.1

5.1 Experimental Setup

In order to evaluate this work in a systematic way, tests were scripted combining sequences of *mechanisms* with *aggressiveness* values, to be employed. The aggressiveness values considered will hold values of 1 and 1024, to assess how the intensification of the limits imposed affects resource usage. In addition, a set of typical web-pages will be used to be rendered in tabs. These comprise web pages of news sites, social networks, sports sites, mail clients and multimedia-streaming sites. This is done to the end of providing a rich and varied web-page suite, for testing.

The main idea of the scripts are to, firstly open a set of pages and secondly navigate through those pages, gathering resource consumption data representing the gains/losses of applying mechanisms on idle tabs, in parallel to page navigation (and considering different aggressiveness values).

For understanding how the employment of certain mechanism combinations might affect Latency, certain browsing habits are simulated through different *tab selection policies*. These policies state what is the next tab to activate (i.e. what page to visualize next). Three policies were used for tab selection,

while navigating tabs:

1. *round-robin selection* to navigate sequentially from tab to tab;

2. *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;

3. *random tab selection* where a certain tab is selected randomly, possibly more than once.

In order to simulate user-interaction with pages, to the end of forcing tabs to consume resources, certain scripted actions took place. One example is to scroll up and down the page, in half-of-a-second intervals (500 ms), each. The time to stay at each web-page is constant for all tab selections. In practise, tabs remained active for 15 seconds, while simulating user-interaction. As a final note, every time a tab is terminated, due to employing *mem hard* or *time hard*, it has its page reloaded once it becomes active again.

Regarding the testing environment, in which the evaluation took place, the specifications are listed as follows:

- The Chrome version was 44.0.2391.0, dev-channel release, 64-bit build. The rendering engine was Blink, build 537.36. JavaScript's V8 engine built-in the browser was in version 4.4.48.1. Flash version was 17.0.0.188. (These data were retrieved by observing the information stored at `chrome://version/`).

- 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 and a Qualcomm Atheros AR9285 Wireless Network Adapter.

5.2 Resource Consumption Analysis

The assessment of resource consumption gains due to the tab extension operation will be done, firstly, in regards to the CPU usage and memory, because those are the types of resources managed directly by the mechanisms under evaluation. The impacts on the energy consumed with data transmissions wirelessly will also be accounted for by measuring tabs' processes bandwidth usage. Since network bandwidth usage is also very much associated to power consumption in devices, some evaluation of the impact and gains on that resource will also be done.

The magnitude, in value, of resource consumption might not always be associated to energy consumption – despite the fact that this seems to be the case for CPU usage and Bandwidth usage (Rodrigues [2013], Bircher and John [2012], Park et al. [2009]). When it comes to memory usage, though,

the variations in memory usage (e.g. due to memory allocations/deallocations) seem to be equally relevant to – if not more than – the amount of memory consumed, since they reflect the inner-workings of system-wide components, (such as CPU instructions execution and disk accesses). Following this rationale, special attention will also be given to the way memory varies, over time, on the analysis ahead.

Resource metric values gathered are not only associated with tab renderer processes, but also with the "kernel" *browser process*, the *gpu process* and the process responsible for running *Flash*. To further account for the overheads inherent to GreenBrowsing functioning, its resource consumption footprint is also included in the resource consumption values gathered.

The infographics shown in the following sections consist of time-series over fixed time periods. Comparison will be made between recorded experiments where no mechanism was active (*all off*) and experiments whose mechanisms were enforced with different aggressiveness values. For all experiments, information regarding the average value recorded for that resource metric will be shown (identified as `average`). The slope of the line obtained through least squares is also presented, to give a sense of how resource consumption evolves over time for each experiment (`growth_rate`). Finally, resource consumption reductions will be computed accounting for the ratio between the total resource consumption recorded (while employing the mechanism) and the resource consumption recorded with no mechanism active will be shown (`reduction`). In detail, resource reduction is computed as

$$1 - \frac{\sum_{i=1}^{t} mechanism\_resource\_consumption_i}{\sum_{i=1}^{t} all\_off\_resource\_consumption_i} \times 100 \tag{5.1}$$

where $mechanism\_resource\_consumption_i$ represents the resource value consumed by a certain mechanism, at instant i and $all\_off\_resource\_consumption_i$ stands for the resource consumption recorded for *all off*, at instant i.

5.2.1 CPU Usage.

When it comes to CPU usage, *cpu* is the only mechanism that acts directly upon the allowed rates for tabs to execute at. It is therefore expected for it to be the one that more accurately can control process usage. On the other hand, mechanisms like *prio* do not directly control any resource metric. As a matter of fact, it does not control any of the resource metrics considered for evaluation. The only expectation is for it to "help" idle tabs, closer to the active tab, to execute more often. Mechanisms *mem* and *time* help reducing CPU usage more substantially (Figures 5.5, 5.3, 5.6, 5.4), since they are not so dependent on the unpredictable behaviour of the Operating System scheduler, as *prio* is.

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 5.1.

When applying *cpu* (Figure 5.2), 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 knowing what the maxi-

mum value for CPU usage will be.

Mechanisms *mem hard* and *time hard* seem to allow the browser to consume between 38% and 45% less CPU, than in the case of *all off*, as show in Figures 5.3 and 5.4.

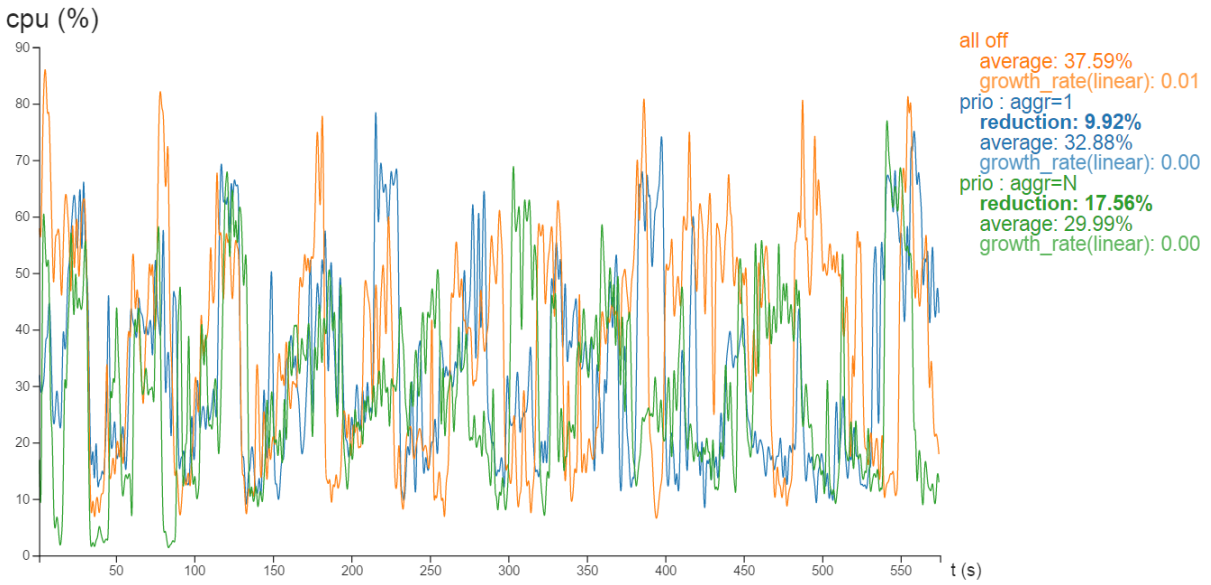


Figure 5.1: CPU usage for *prio*.

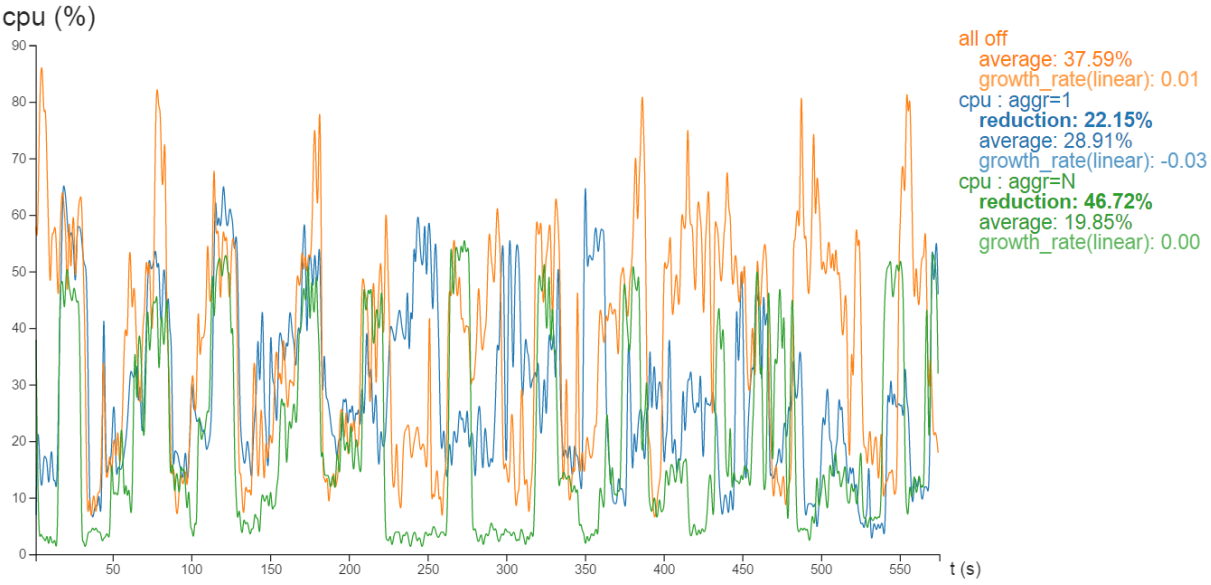


Figure 5.2: CPU usage for *cpu*.

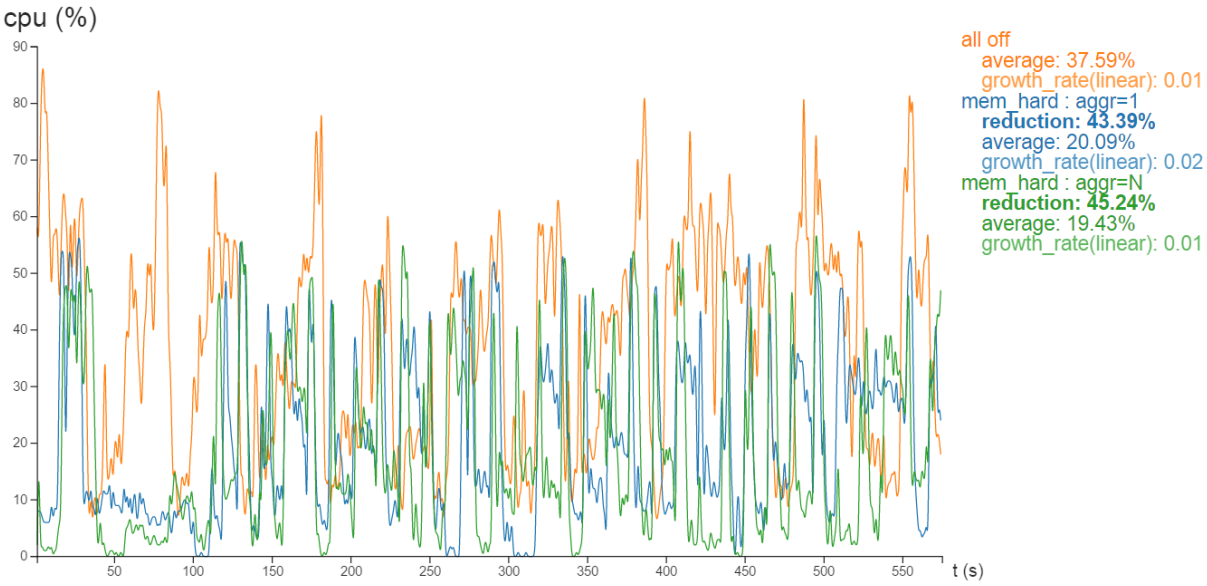


Figure 5.3: CPU usage for *mem hard*.

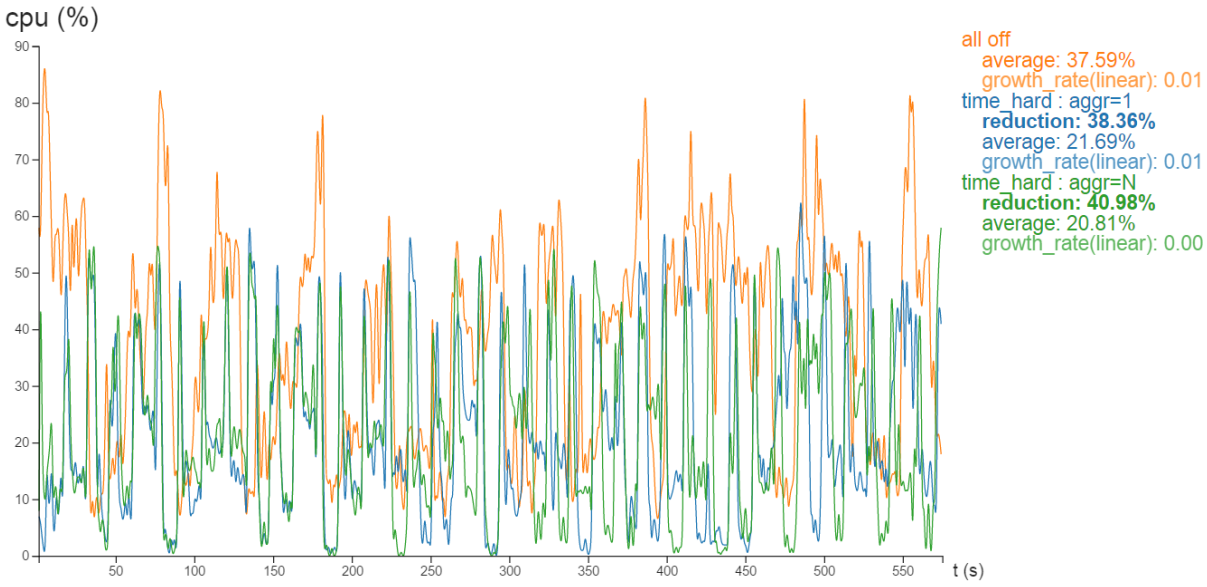


Figure 5.4: CPU usage for *time hard*.

Both *mem soft* and *time soft* (Figures 5.5 and 5.6 respectively) hold the best results for CPU usage reduction, with decreases of up to 80% when applying *mem soft*. This is justified by having less tab processes executing in parallel. Both *mem hard* and *time hard* achieve minor reductions than their *soft*er versions. This is explained, in part, by the constant page reloading and re-rendering, over time.

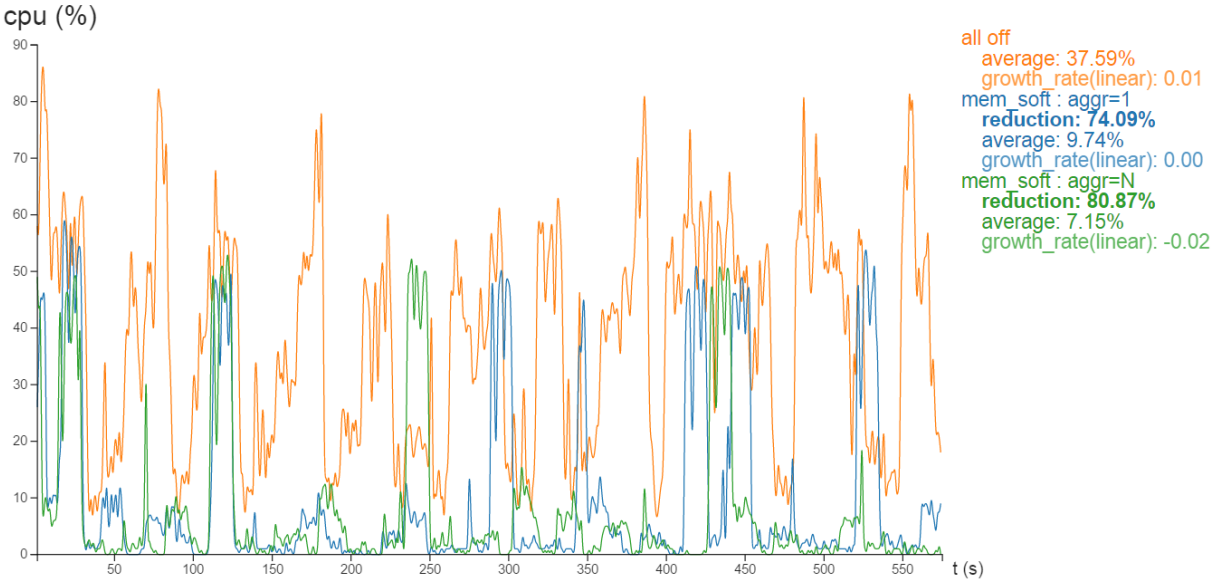


Figure 5.5: CPU usage for *mem soft*.

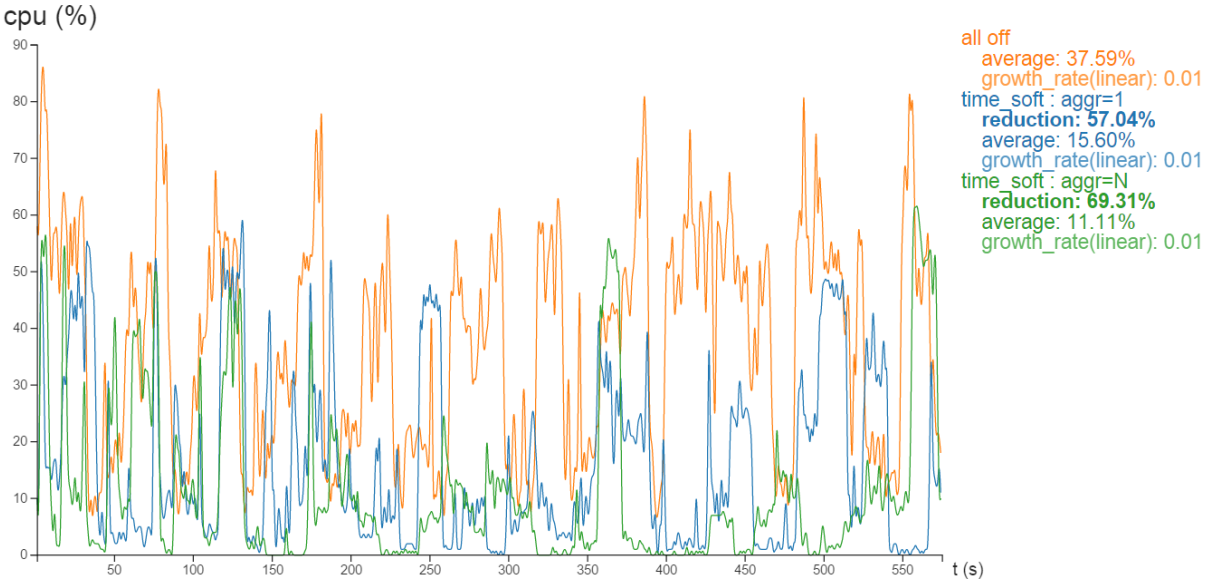


Figure 5.6: CPU usage for *time soft*.

5.2.2 Memory Usage.

The mechanisms that adjust memory directly are *mem soft* and *mem hard*, so it is interesting to see how their functioning affects memory usage. But it is also interesting to see how the application of different mechanisms affects memory usage, in particular if they are able to tame memory variations over time. In this section, we are interested in assessing what memory usage and variation mitigation results from the application of the different mechanisms available.

Both *prio* and *cpu* present highly subjective and dubious reductions (Figures 5.7 and 5.8): when comparing all series, they start off from different memory values, hence these reductions are not due to

the mechanisms at hand. Further, each of the mechanism's data paths seems to grow more rapidly than the one of *all off*, for the two values of aggressiveness considered. Hence, any conclusion regarding the effectiveness of *prio* and *cpu*, in reducing memory usage, should be taken lightly, when considering these results.

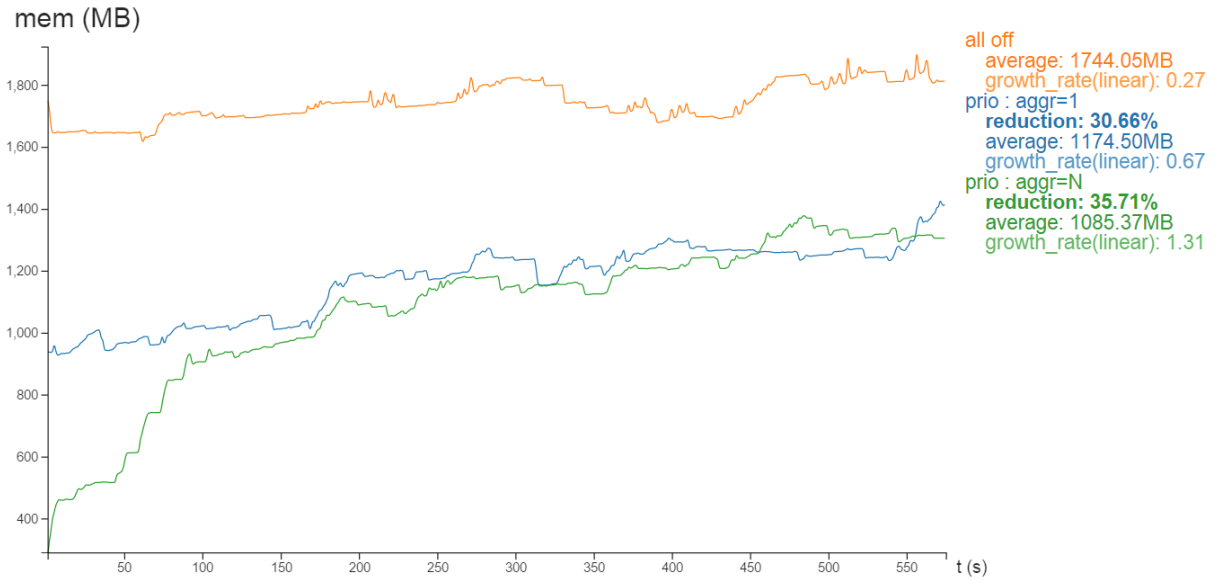


Figure 5.7: Memory usage for *prio*.

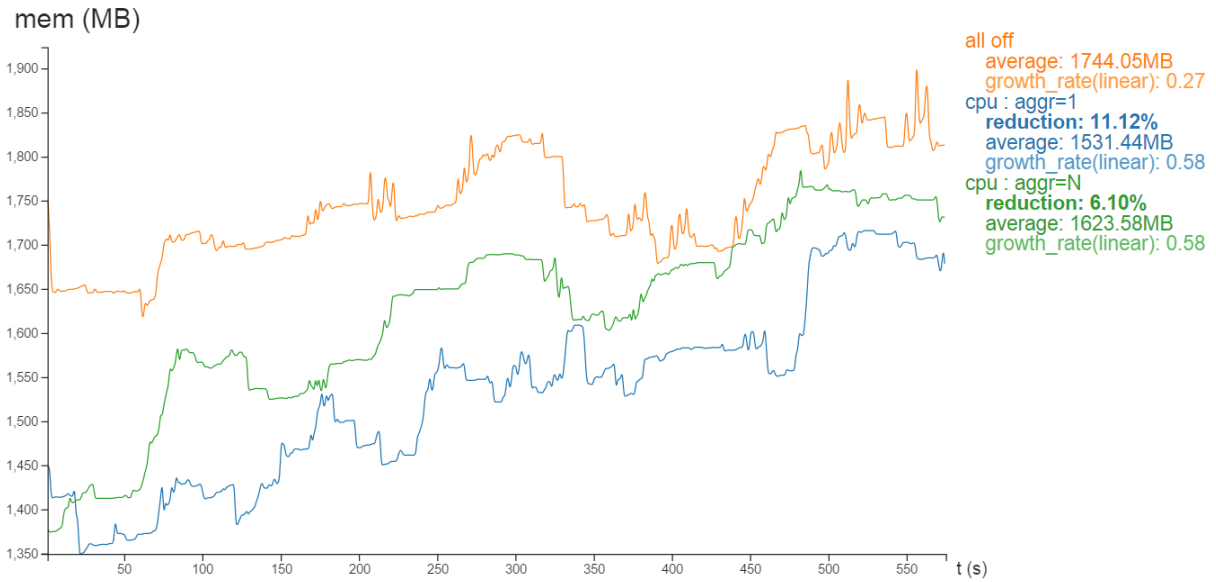


Figure 5.8: Memory usage for *cpu*.

Figures 5.9 and 5.10 demonstrate how *mem* and *time* evolve over time, when compared with *all off*. The first thing that is noticeable is how evident memory reductions are for *mem hard* and *time hard*. Another thing to notice is that *time hard* seems to reduce memory usage faster than *mem hard*, since the first took approximately 100 seconds until reaching smaller memory ranges, while the latter induced

smaller memory values in about 10 seconds, after starting testing. Finally, looking at *mem soft* and *time soft* data paths, we can deduce they are less prone to vary, over time, than in the case of *all off*. And even less than their *hard* counterparts. Once again, the memory reductions are subjective to the starting memory values, that the browser entities under testing were consuming, at the time each mechanism was run. Two additional time-series can be observed in Figures 5.11 and 5.12.

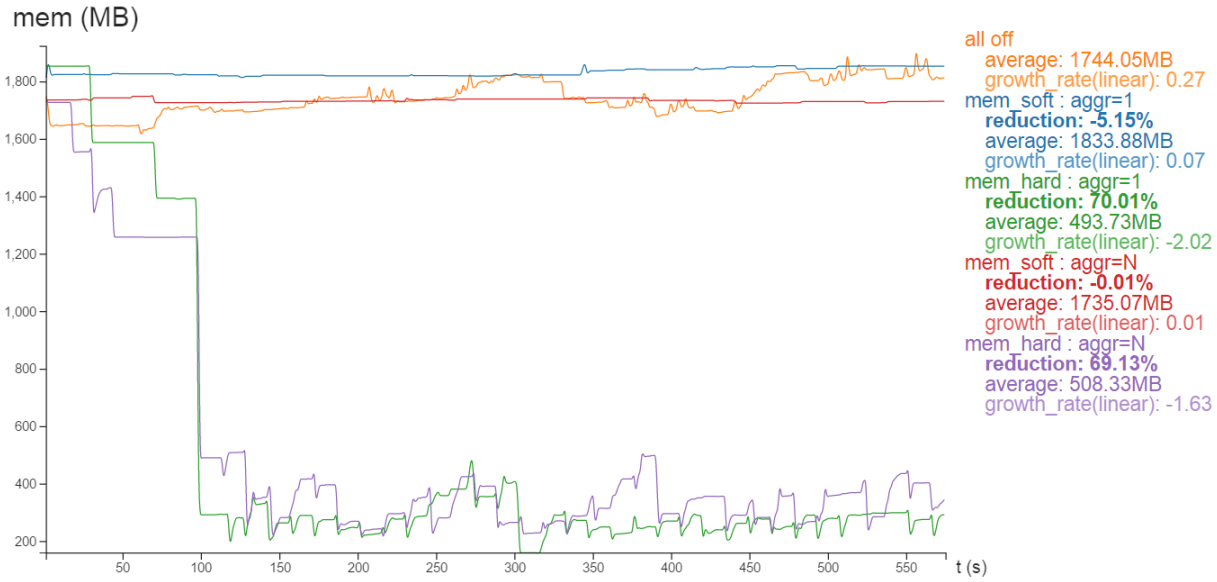


Figure 5.9: Memory usage for *mem soft* & *mem hard* – initial values.

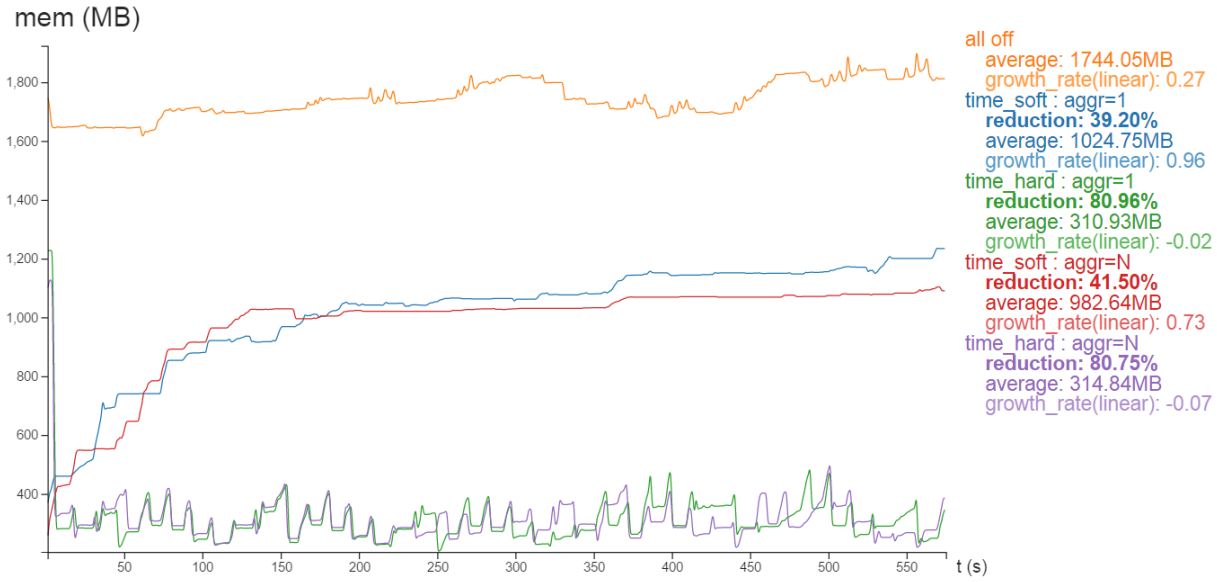


Figure 5.10: Memory usage for *time soft* & *time hard* – initial values.

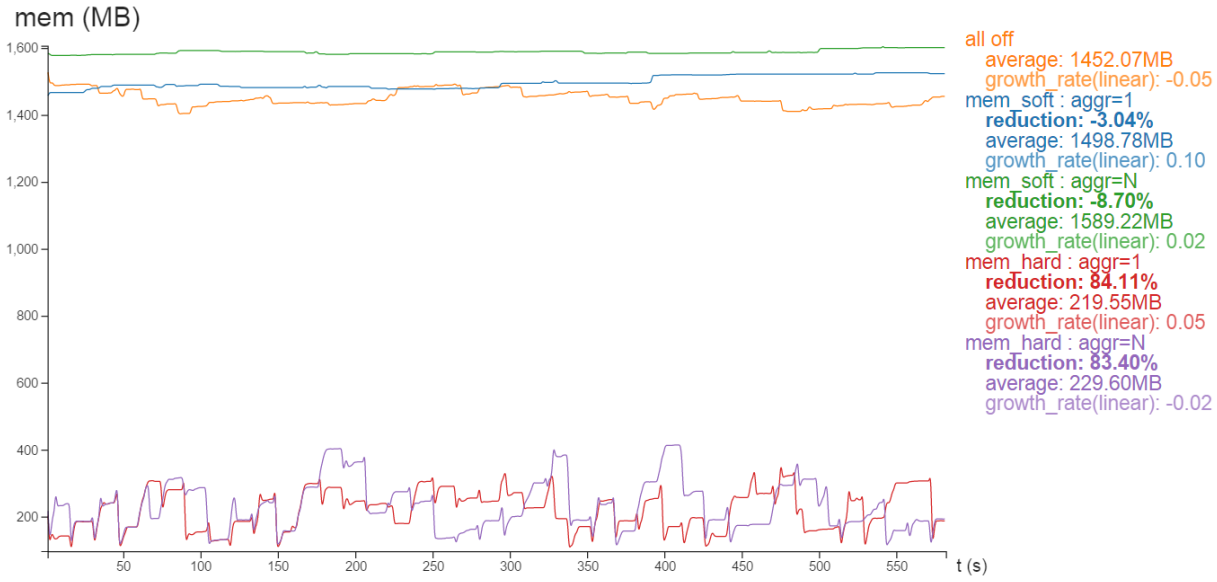


Figure 5.11: Memory usage for *mem soft* & *mem hard*.

This time, we allowed the experiments to run longer, (twice in duration), in order to disregard the initial stages, accounting just for the periods where the memory variations were more stable, varying in smaller memory ranges. This way, memory gains can be more accurately and objectively compared.

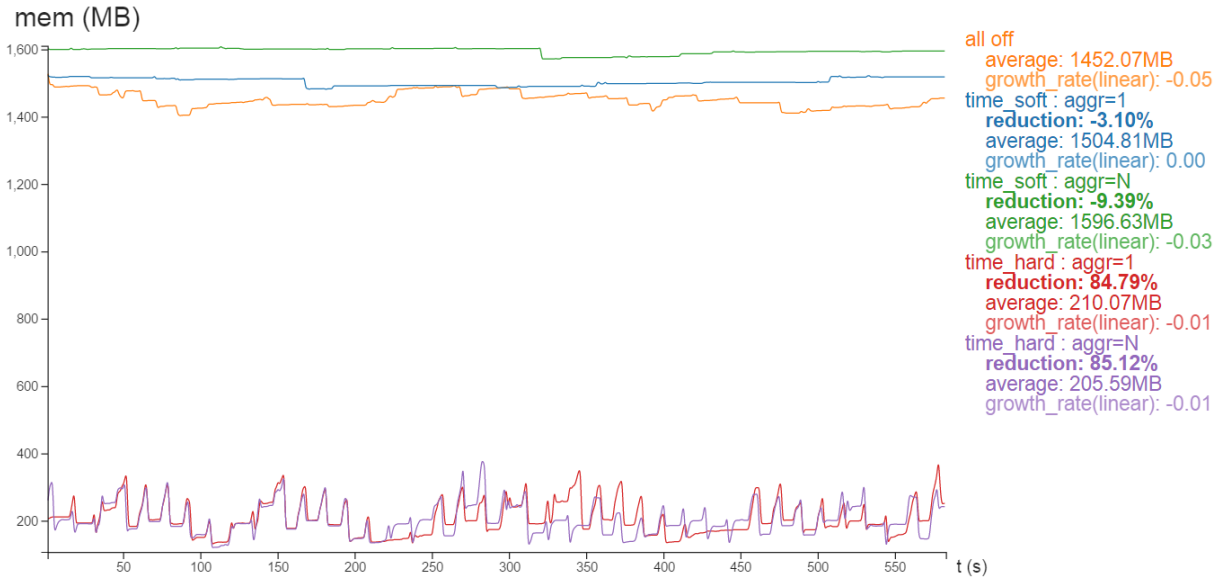


Figure 5.12: Memory usage for *time soft* & *time hard*.

5.2.3 Network Bandwidth Usage.

No mechanism adjusts bandwidth usage directly, but there is a chance for all of them to influence it. This can be achieved either by, for instance, adjusting cpu rates with mechanism *cpu*, or by halting process execution with *mem soft*. These are the results expected of the network bandwidth usage benchmark.

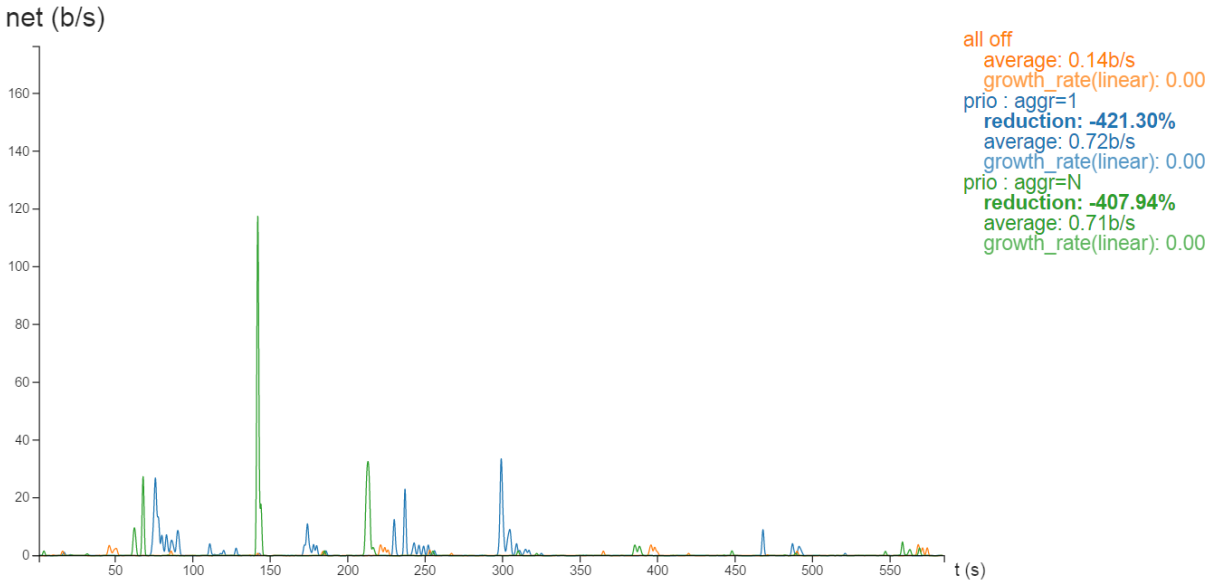


Figure 5.13: Bandwidth usage for *prio*.

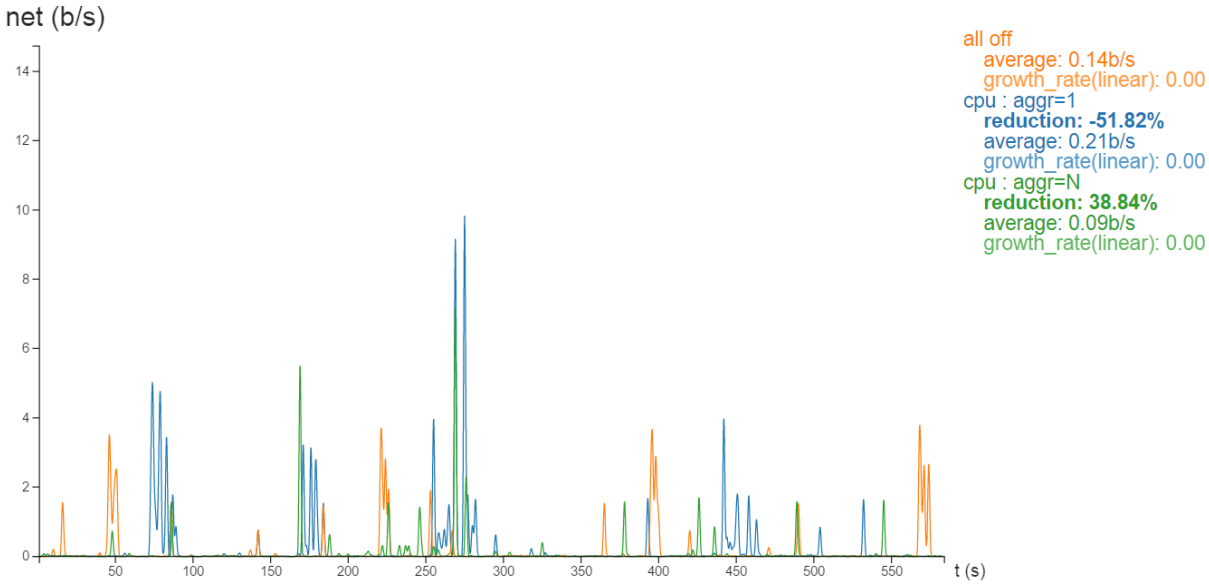


Figure 5.14: Bandwidth usage for *cpu*.

While employing *prio* there are evident increases in bandwidth usage, compared to no mechanism in use. It is not perfectly clear, though, if those increases are due to the effects of using *prio* or if those increases are due to random data transfers that happened while browsing. The latter seems to justify in part, these increases, since bandwidth usage spikes can be identified in specific moments, over time. For example, between 140 and 150 seconds, after starting the experiments (Figure 5.13). But, by exclusively taking into account results recorded in experiments, *prio* is concluded to be of no use to control bandwidth usage.

Mechanism *cpu* is not exactly en-par with *prio*, since there are recorded reductions in bandwidth

usage for high values of aggressiveness. It is not clear, however, how could the act of employing an aggressiveness of value 1 induce higher bandwidth usage values.

Both *time soft* and *mem soft* seem to influence reductions in bandwidth usage. This might be justified by the nature of the mechanisms, since they consist in halting tab process execution of idle tabs. If no tab process transmits data in the background, while the tab is idle (being logically only allowed to transfer while the tab's process is executing – and the tab is active) then bandwidth usage will decrease due to having less data transfers happening at each time.

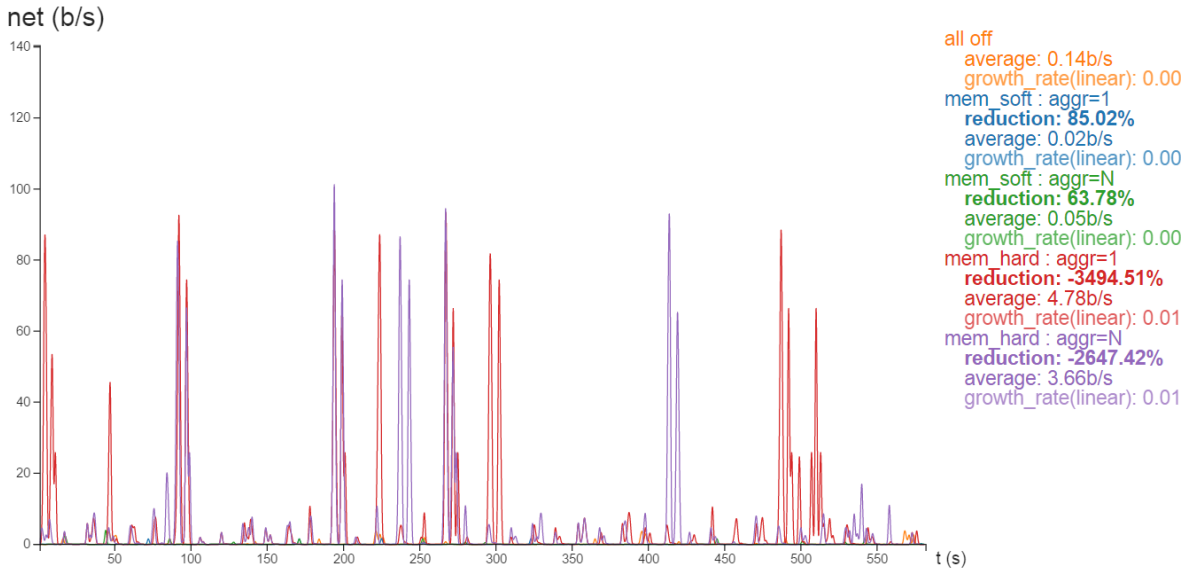


Figure 5.15: Bandwidth usage for *mem soft* & *mem hard*.

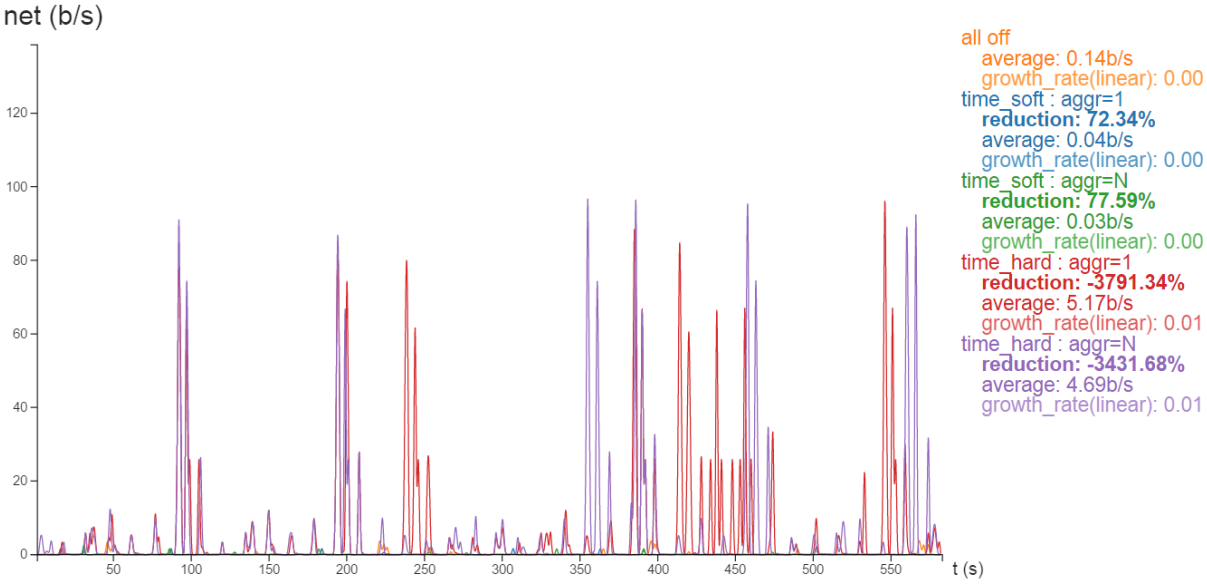


Figure 5.16: Bandwidth usage for *time soft* & *time hard*.

On the other hand, *mem hard* and *time hard* seem to be devastating in terms of bandwidth usage.

This is expected, since tabs re-load each time they are re-activated, sometime after after having their processes terminated due to limit violation.

5.2.4 Discussion.

Mechanism *prio* is the most unreliable of the mechanisms presented. Even though the results show some reduction when employing prio, specifically in CPU usage, its effects are not as predictable as the effects of other mechanisms. If it is considered that there are more applications running in parallel with the browser, each idle tab's process is actually competing with all the user-owned processes in execution, at that time. This could be advantageous for the purpose of reducing resource consumption if the priorities, that other processes operate on, are superior to the ones of idle tabs. But if they are not, the reductions might not be as perceptible as in the previous case, since similar valued priorities probably result in similar processor time. Of course, these claims only make sense if the Operating System's scheduler, indeed, takes priority scheduling in account to some extent. Otherwise, there is no real advantage in applying *prio*.

When it comes to *cpu*, resource usage patterns reflect the adjustments made, better. Specifically in the case of CPU usage. If the latter is the only/main metric that is wished to be adjusted, this mechanism should be preferred to *prio*.

Even though *mem* and *time* offer similar behaviour with their *soft* and *hard* versions, there are some scenarios where *time* could be chosen over *mem*. In particular, if the hard version is considered. Comparing Figures 5.9 and 5.10, it is possible to observe that memory usage decreases much more rapidly in the *time hard* experiment, holding increased reductions in memory usage. For memory-constrained devices, this might be of use, since smaller memory usage ranges are induced faster.

There are reasons to disregard *mem hard* and *time hard*, though, if it is intended to reduce energy consumption above all other constraints. The main reason is due to the smaller values of memory being achieved at the expense of highly disproportional network usage. One could argue that the increases in network transmissions might be explained by the testing behaviour itself, since each time a tab whose process was terminated is re-activated, a new process is created and the web-page is fetched again (possibly from a remote location). But this testing behaviour surely coincides with what some (if not most) users do: reload the page if the tab has ”crashed”. Therefore, the energetic concerns that arise from these situations, where *mem hard*/*time hard* are employed, come from the fact that network data transfers are highly associated with power consumption, specially in the case of wireless transmissions [Balasubramanian et al., 2009]. The conclusion is that *mem hard* and *time hard* proved to be counter-productive, in terms of energetic reductions, when used in similar situations to the testing experiments.

Reloading tabs also leads to some CPU usage, since web-pages fetched after a new tab is created, in-place of one that was terminated, need to be re-rendered. Be that as it may, experiments show substantial reductions in CPU usage, still, by employing *mem hard* and *time hard*; *mem soft* and *time soft* achieved better results than their counterparts, though.

A final remark on the observations done for *mem hard* and *time hard* is that they provoke the biggest variations in memory consumption. Even though these variations occur at low ranges of values, their significance should not be overlooked, since memory variations might be an sign of system-wide activity, which will incur in additional energy consumption.

Regarding *mem soft* and *time soft*, they seem to be the ones that provided the most substantial CPU usage reductions, lesser memory variations (even though not reducing process private memory in some cases), and the only ones to reduce bandwidth usage, all-together. Having this into account, they seem to be the ones that contribute to more prominent energy reductions, for situations similar to the testing methodology described at the beginning of Chapter 5.2.

Since in the great majority of experiments resource consumption values were decreased, in comparison to not using the extension at all, it can be concluded that GreenBrowsing effectively reduces resource usage and, with the exception of *hard* mechanisms, the resulting energy footprint of browsing the web.

5.3 User Experience Evaluation

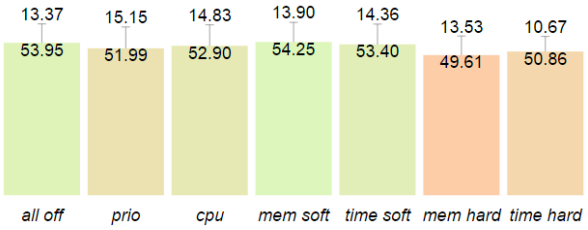
In order to assess what are the implications in terms of user-experience significant requirements, Frames-per-second (FPS) and Latencies (as defined in Section 5.1) were recorded, while running resource consumption tests.

FPS were considered since it might be possible for the resource reduction mechanisms to reduce the rate at which animations, and other graphical entities, are drawn. If the reductions in FPS are too strict, stuttering visuals might be perceived by users, which would degrade the overall browsing experience.

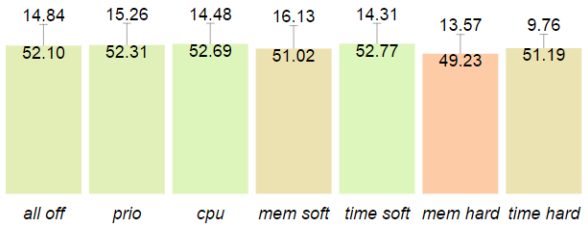
By analysing Latency – time spent in tabs loading their pages, after consecutive tab selections – it is possible to have an idea of the user-perceived web-page processing delays introduced by the application of certain mechanisms. Latency measurements will be distinguished based on tab selection policies. The idea is to see if the way of selecting tabs can also induce further experienced latencies. This will also be done when considering FPS measurements.

5.3.1 Frame-rate Accounting.

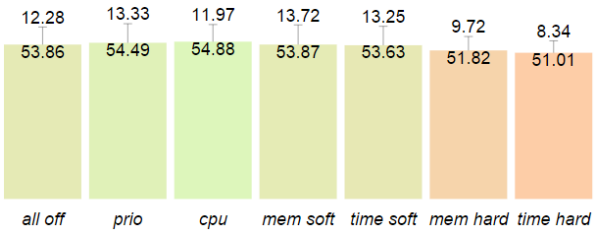
Figure 5.17 shows the frame-rates recorded for tabs that were active, for each mechanism employed and while selecting tabs according to the three selection policies described previously. Bars amount to average FPS recorded, while the vertical lines embody the standard deviations of the samples gathered, for each mechanism. Greener, taller bars represent better frame-rates, while redder ones symbolize worse frame-rates.



(a) FPS measurement / round-robin tab selection



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



(c) FPS measurement / random tab selection

Figure 5.17: FPS measurements for the 3 tab selection policies considered.

In general, the frame-rates recorded are around 50 FPS. Even for *hard* mechanisms, the reductions were just slightly smaller than those of other mechanisms.

Typically, pages achieve frame rates of 60 FPS, during most of the time they are visualized. However, every time a new tab is selected, frame-rates drop for some moments, no matter the mechanism employed – even in the case when mechanisms are *all off*. This explains the averages recorded being a bit lower than what was possibly expected.

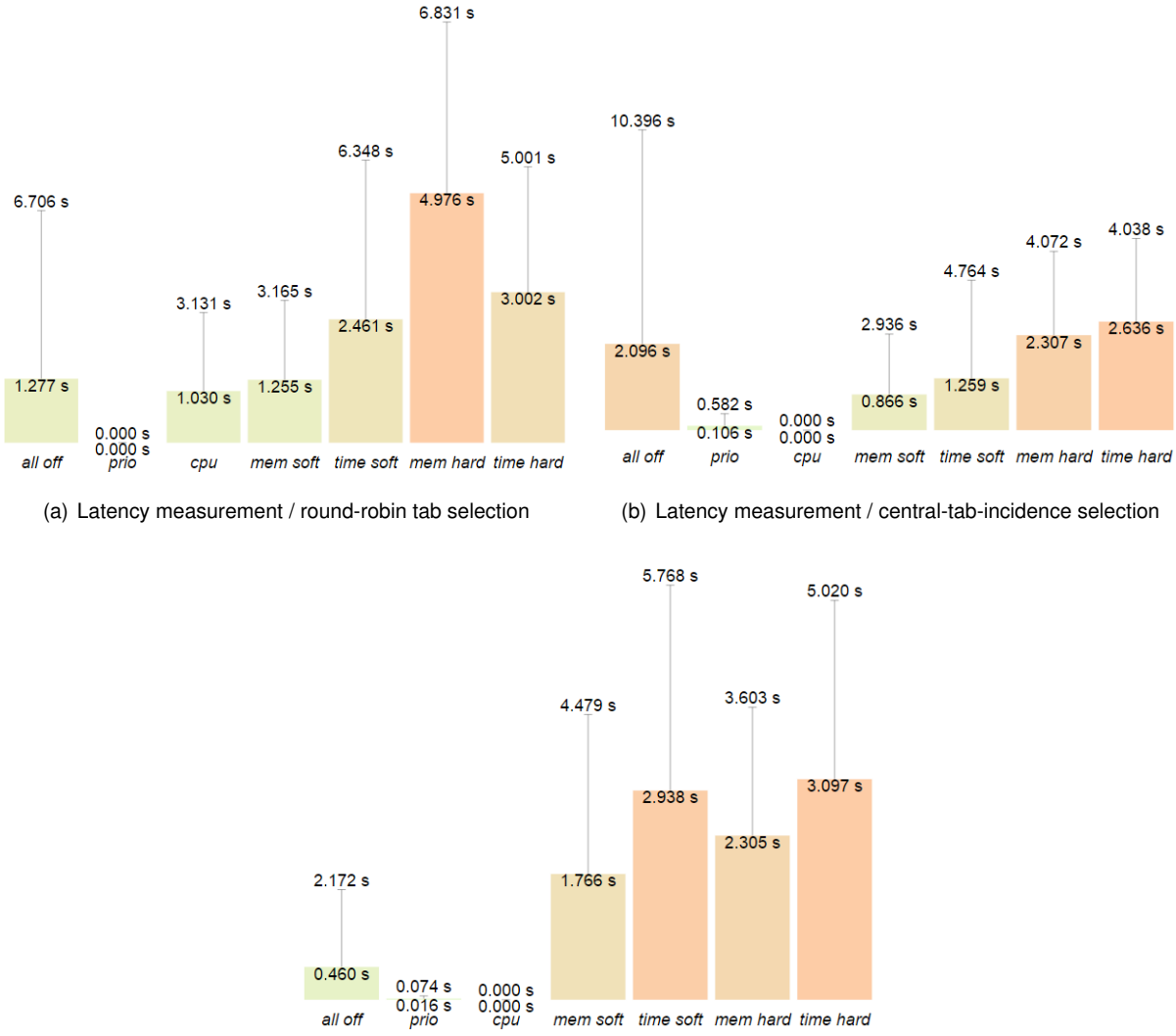
Therefore, applying resource reduction mechanisms does not seem to incur in noticeable user-experience degradation, when it comes to frame-rates. And it should not be any, since mechanisms are employed exclusively on idle tabs, that are not visualized by users.

5.3.2 Latencies per-Mechanism.

Figure 5.18 presents the latencies experienced, as rectangles and on average, for each tab selection policy. Standard deviations of latency periods are depicted as whisker-like vertical lines above histogram rectangles.

What is possible to see is 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. This might be explained in part by the Last-Time-usage and Active-Tab-Distance assumptions (introduced in Section 3.1.3) because by selecting a tab randomly, there is a chance of selecting a tab is idle for a long time or that is rather far from the active tab. Since those tabs tend to be acted upon sooner than others, closer to the active tab or that were active more recently, chances are for tabs pages to be re-loaded

due to their process being halted and having received data that was awaiting for it, to execute again.



(a) Latency measurement / round-robin tab selection

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

(c) Latency measurement / random tab selection

Figure 5.18: Latency measurements for the 3 tab selection policies considered.

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). These occasional latency periods did, therefore, increase the latency averages documented.

5.3.3 Resource Consumption and Latency.

In order to have an idea of how Resource Consumption data match Latency records, the inspection of *correlative relations* between these two dimensions was done, for both *soft* and *hard* versions of *mem* and *time* mechanisms. They were chosen because the biggest latency values were recorded while employing them.

The importance of assessing how resource consumption correlates (or not) with latency is in giving insight of what factors might condition that correlation (or anti-correlation). This will be done in an ex-

ploratory sense, not saying anything regarding causality relations that might occur between the random variables, under testing. In this way, the analysis will be merely speculative, with factual-based reasoning nonetheless.

Latency values correspond to time periods in which many resource consumption values might be gathered. Considering, for instance, the record of a latency period lasting of 6.5 seconds, 6 CPU values could be observed, since resource consumption measurements are done once per-second.

In order to avoid the non-correlative influence of having multiple observations of one random variable for a single observation of the other, one-to-one observations are enforced, for each random variable, when testing correlation relationships. This is done independently for each random variable data set, according to the following formulas:

- When considering n CPU values in a single latency period, the CPU usage (in percentage of processor time) associated with the latency time interval is given by:

$$cpu\_usage = \sum_{i=1}^{n} \frac{cpu_i}{n}$$

- In the case of n memory values, the difference between the maximum and the minimum recorded values is considered, hence:

$$\Delta memory = memory_{max} - memory_{min}$$

  This is done, in order to be conforming with the assumption ”memory variations represent system-wide behaviour and therefore power consumption”.

- When it comes to bandwidth usage, the total bits transferred, during latency periods, are considered instead:

$$datatransferred = \sum_{i=1}^{n} bandwidth\_usage_i$$

  This is done because wireless data transfers are considered to be highly related with energy dissipation in devices [Balasubramanian et al., 2009].

Since time intervals and, in general, range of values are being considered, the random variable's observations at hand are interval-scaled. Pearson's correlation method along with Kendall's correlation method are used.

Figure 5.19 presents the correlative relations for the three resource consumptions metrics in respect to latency. These results will be used as reference when assessing relations of other mechanisms with latency.

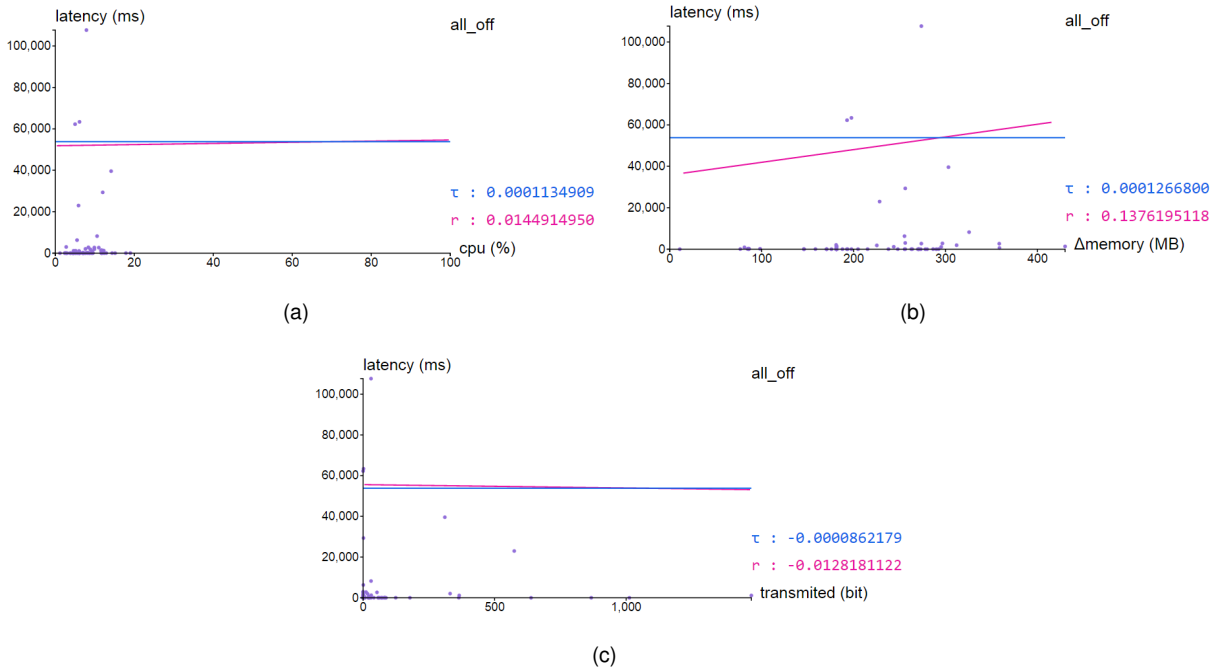


Figure 5.19: Resource consumption and latency correlation for all off for *all off*.

Figures 5.20 and 5.21 show how CPU usage behaves in the presence of latency periods, while applying *mem* and *time*.

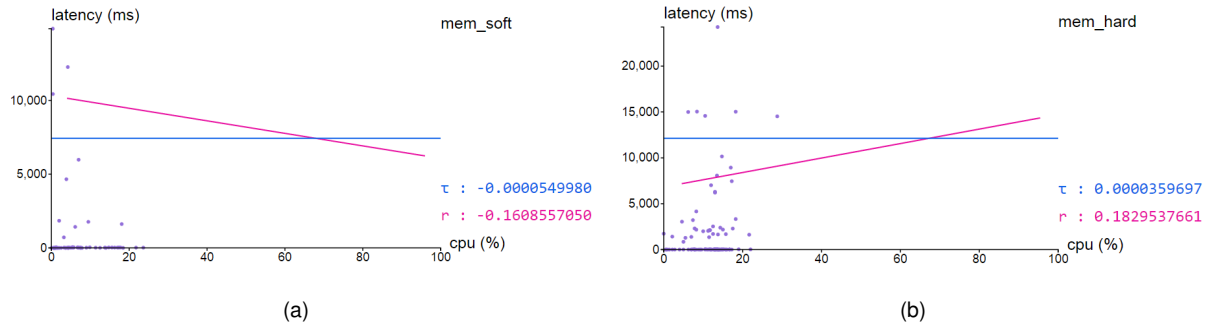


Figure 5.20: CPU usage and Latency correlation for *mem soft* and *mem hard*.

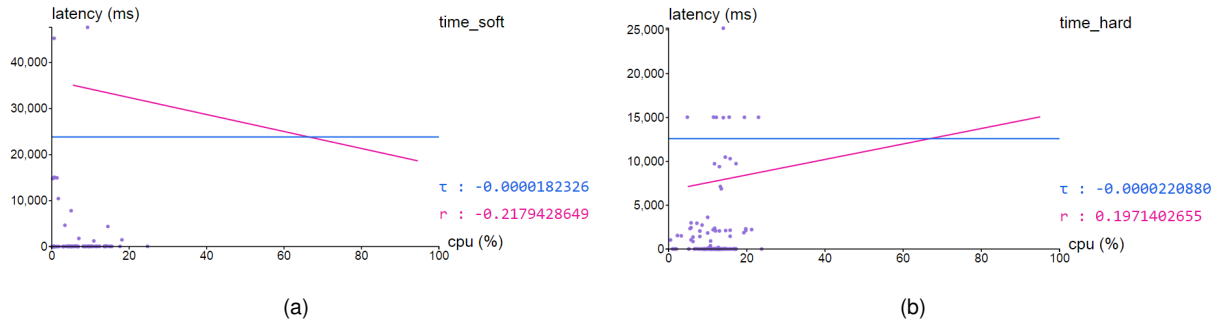


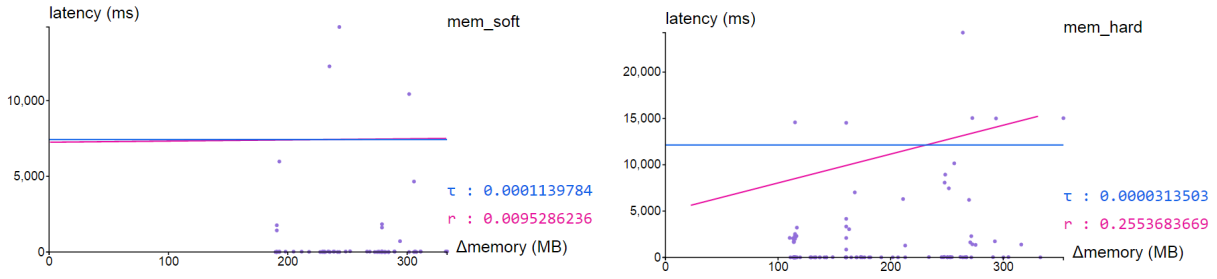
Figure 5.21: CPU usage and Latency correlation for *time soft* and *time hard*.

For both *mem hard* and *time hard* CPU usage seems to be slightly correlated with latency. This may be due to the fact that pages (in tabs whose process was terminated) reload each time their tabs are re-

activated, according to the testing methodology followed. On the other hand, on the *soft*er mechanism versions, CPU usage seems to faintly anti-correlate with Latency. CPU usage does not seem to be correlated with Latency while mechanisms are *all off*, though.

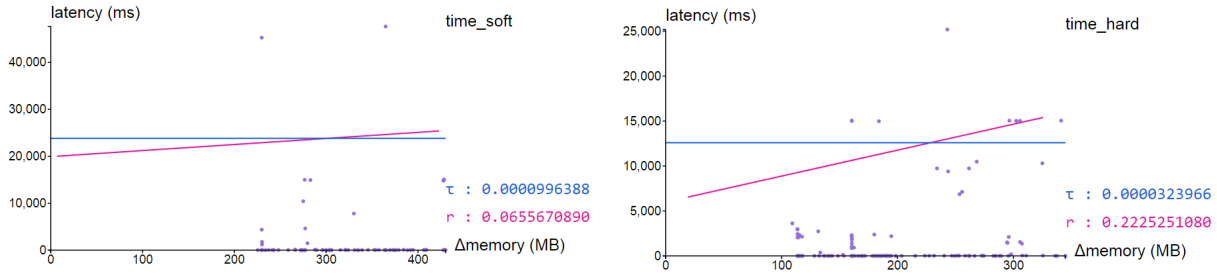
It would be, nevertheless, interesting to understand why such relations hold when applying *soft* mechanisms. We can hypothesize that since idle tabs use less computational resources over time (since after becoming idle, halting a certain tab's process is imminent), the browser is able to make better use of CPU, because the active tab process would be competing with fewer processes, for processor time, than it otherwise would if mechanisms were *all off*.

When looking at Figures 5.22 and 5.23 no significant correlation values are found between memory variations and latency experienced when considering *soft* mechanism. Their *hard* counter-parts on the other hand, are again correlated with memory variations. The assumptions made for CPU usage are the same for memory variation, in the case of *mem hard* and *time hard*.



(a) (b)

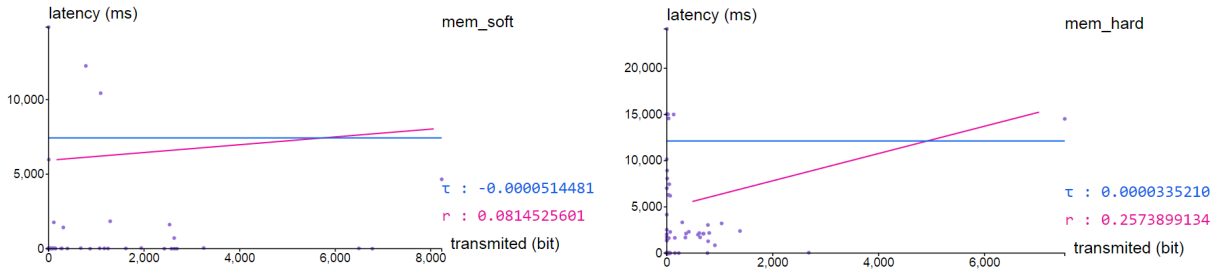
Figure 5.22: Memory variation and Latency correlation for *mem soft* and *mem hard*.



(a) (b)

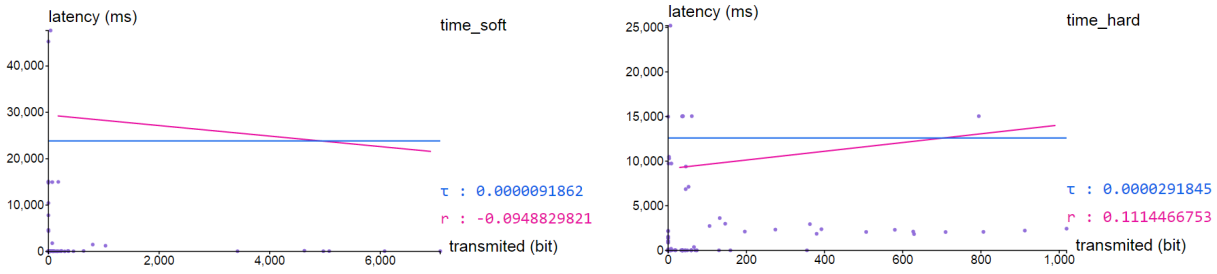
Figure 5.23: Memory variation and Latency correlation for *time soft* and *time hard*.

The amount of data transferred for each recorded latency period, while employing *mem* and *time* is depicted in Figures 5.24 and 5.25, respectively. In these experiments, the amount of data transferred seems to correlate positively with the latency experienced, in the *hard* cases. This is probably due to these transfers being associated with data being received/fetched from the web, hence allowing us to conjecture that the higher the quantity transferred, the bigger the latency period. The positive and negative correlations for *mem soft* and *time soft*, respectively, seem rather insignificant to be given importance.



(a) (b)

Figure 5.24: Bandwidth usage and Latency correlation for *mem soft* and *mem hard*.



(a) (b)

Figure 5.25: Bandwidth usage and Latency correlation for *time soft* and *time hard*.

It is possible to observe that occasional latency values greatly exceed – in value – the great majority of latency records, when considering just one mechanism. An example is *mem soft*, as depicted in Figure 5.24. This partially explains why the standard deviation values were so high.

As a final note, the correlation values were, in general, too small to be considered relevant. Even though this seemed a fitting analysis for the case of assessing if there was, at least, a chance of lower resource consumption rates inducing higher latencies or the other way around.

5.4 Evaluation Summary & Conclusions

In this Chapter, the evaluation of GreenBrowsing took place in regards to its effectiveness in reducing resource consumption, assumed energy consumption and latencies introduced by applying mechanisms. While no mechanism appears to be sufficient to reduce CPU usage, memory usage, variation of memory and (to a smaller extent) bandwidth usage all at once, *mem soft* and *time soft* were the mechanisms that reduced the most CPU and bandwidth usage, while slowing the pace at which memory varied, over time.

When evaluating the Extension in regards to latency, latencies recorded for sequential tab selections were, typically, smaller than in the case of tabs selected randomly. This might be explained partially due to the inner-workings of the Extension, since it functions by enforcing the assumptions of Last-Time-Usage and Active-Tab-Distance (as described in Section 3.1.3). It was possible to observe that data transfers were somewhat correlated with the latencies observed, which allow us to speculate that

latencies were induced while transferring data, possibly due to pages being forced to be reloaded, especially in the cases of *mem hard* and *time hard*. It was also possible to observe that the greater the memory variation and the CPU usage, the smaller the corresponding latency period would be, when considering *soft* mechanisms. This allowed us to conjecture that because resource usage is lessened for idle tabs, (since their processes were halted), active tabs' processes are free to use more CPU to achieve better levels of performance. It can also mean that some additional latencies are introduced when applying soft mechanisms, (lower resource consumption rates associated with higher latency values). Whichever the case, correlations are, probably, too low to support any of the previous suppositions.

Chapter 6

Conclusions & Future Work

6.1 Conclusions

In this work, GreenBrowsing was presented as a system comprised of:

- a Google Chrome Extension, used to reduce consumption and the inherent energy footprint of computational resource, and of:
- a Back End subsystem, that certifies URLs' and domains' web-pages energetically-wise, in regard to the resource consumption introduced in processing them.

In Chapter 1 the problems related to energy-inefficiencies in computing, as well as the problem of energy-intensive browsers, were presented as a motivational factor to this work. Challenges for this work and shortcomings of some browser power-management solutions were discussed.

In Chapter 2 both seminal and recent body of work, related to the objective of minimizing power consumption at different levels was presented. Some insight, on what requirements are important for processing energy data, was also taken from the study done on relevant systems.

In Chapter 3, the architecture of both the Extension and the Back End subsystems were described, in a top-down approach. The chapter starts by introducing the modules and run-time components that compose the Extension, mapping them to their respective functionality. After that, the methods used to manage resource consumption through tabs' processes are explained. At this point, the mechanisms employed to achieve resource reduction were detailed. After that, the components of the Certification Back End were exposed, emphasizing the flexibility achieved by separating concerns associated with each component. The distinction between modelling certification ranks and certifying URLs and domains, themselves, was done since those are two separate stages in the certification process. The algorithms used in those stages were explained, detailing the requirements for devising a certification scheme and the statistical methods employed.

In Chapter 4 the implementation details regarding each of GreenBrowsing's subsystems were presented. The chapter starts with the exposition of the two components that comprise the Extension, namely the component that runs in the browser as a Google Chrome Extension and the auxiliary Back-

ground Process that runs natively (as a Windows application) and that receives browser state information, in order to act on tab processes. The problems that arose from the asynchronous nature of Chrome events, namely the care that needed to be given to the way those events were dealt with in the Background Process, were presented together with their solution. An overview of the Chrome Sandbox Model on Windows was done in order to understand what entities needed to be managed – Job Objects – and also how the resource reduction mechanisms could be enforced over tab processes, using Job Objects. Lastly, the details about the technological choices for the Back End, in terms of auxiliary frameworks and libraries, were given.

At Chapter 5 the evaluation of GreenBrowsing was done in respect to resource reductions achieved when employing each mechanism and the user-perceived delays induced by each mechanism. The chapter ends with the correlative analysis between Resource Consumption and Latency.

The main contributions of this work are, therefore, a tab-management solution (implemented as a Google Chrome extension) and an energy-related certification scheme implemented on a separate subsystem, to be deployed remotely, (e.g. on a cloud setting). Evaluation shows *substantial resource usage reductions*, on energy consumption-related resource metrics (up to 80% for CPU, 85% for memory usage and 85% for bandwidth usage) while *preserving acceptable user-perceived delays* (unnoticeable in most cases). All of this when comparing GreenBrowsing-aided web-navigations with standard navigations.

6.2 Future Work

In this work, the mechanisms presented acted mostly upon CPU, memory and marginally upon bandwidth usage – actually some mechanisms proved to not be suitable for situations were bandwidth usage might be the one of the main factors in inducing additional energy consumption. In this way, it would be interesting to augment GreenBrowsing in order to account for the energetic inefficiencies related to network data transfers.

The Back End could use some extra work when it comes to managing resource usage reads and writes in the Data Store, in order to make it scale better with the number of certification requests. The Certification Modeller could be more fit to process large data sets, by enabling a streaming behaviour, where resource records could be streamed from the Data Store, to the Certification Modeller, where they would be written to disk right-away, in a pipelined manner.

These additions would prove important, since one trend regarding computing seems to lie in the resource-intensiveness and, therefore, energy-demand of systems. To give users information regarding the energetic impact their favourite pages cause and the chance to control how much energy their browsing habits induce feels liberator and right. Users should be able to, at least, adjust performance needs in favour of energy-efficiency, or the other way around, depending on the context of their browsing activities – work or leisure, for instance. In the end, everybody wins: both Users and the Planet, alike.

Bibliography

In *AMD PowerNow!™ Technology*. 2000. URL `http://support.amd.com/TechDocs/24404a.pdf`. accessed: 2015-05-05.

N. Amsel and B. Tomlinson. Green tracker: A tool for estimating the energy consumption of software. In *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '10, New York, NY, USA, 2010. ACM.

N. Amsel, Z. Ibrahim, A. Malik, and B. Tomlinson. Toward sustainable software engineering (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 976–979, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0445-0.

B. Balaji, H. Teraoka, R. Gupta, and Y. Agarwal. Zonepac: Zonal power estimation and control via hvac metering and occupant feedback. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, BuildSys'13, New York, NY, USA, 2013. ACM.

N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, New York, NY, USA, 2009. ACM.

L. Benini, A. Bogliolo, S. Cavallucci, and B. Riccó. Monitoring system activity for os-directed dynamic power management. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, ISLPED '98, New York, NY, USA, 1998. ACM.

A. P. Bianzino, A. K. Raju, and D. Rossi. Greening the internet: Measuring web power consumption. *IT Professional*, 13, 2011.

W. L. Bircher and L. K. John. Complete system power estimation using processor performance events. *IEEE Transactions on Computers*, 61(4):563–577, 2012.

F. Camps. Web browser energy consumption. 2010.

M. Chetty, A. B. Brush, B. R. Meyers, and P. Johns. It's not easy being green: Understanding home computer power management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09. ACM, 2009.

A. K. Datta and R. Patel. Cpu scheduling for power/energy management on multicore processors using cache miss and context switch data. *IEEE Transactions on Parallel and Distributed Systems*, 2013.

J. N. de Oliveira e Silva, L. Veiga, and P. Ferreira. SPADE: scheduler for parallel and distributed execution from mobile devices. In *ACM/IFIP/USENIX 9th International Middleware Conference (6th International Workshop on Middleware for Pervasive and Ad-hoc Computing - MPAC 2008)*. ACM, Dec. 2008.

C. de Siebra, P. Costa, R. Marques, A. L. M. Santos, and F. Q. B. da Silva. Towards a green mobile development and certification. IEEE, 2011.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of The Royal Statistical Society, Series B*, 39(1):1–38, 1977.

S. Esteves, J. N. de Oliveira e Silva, and L. Veiga. Quality-of-Service for consistency of data geo-replication in cloud computing. In *International European Conference on Parallel and Distributed Computing (Euro-Par 2012)*. Springer, LNCS, Aug. 2012.

S. Esteves, J. N. de Oliveira e Silva, and L. Veiga. Flux: a quality-driven dataflow model for data intensive computing. *Journal of Internet Services and Applications (JISA)*, 4(12):1–23, Apr. 2013.

S. Esteves, J. N. de Oliveira e Silva, J. P. Carvalho, and L. Veiga. Incremental dataflow execution, resource efficiency and probabilistic guarantees with fuzzy boolean nets. *Journal of Parallel and Distributed Computing (JPDC)*, 2014.

D. Garlan, F. Bachmann, J. Ivers, J. Stafford, L. Bass, P. Clements, and P. Merson. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, 2nd edition, 2010. ISBN 0321552687, 9780321552686.

M. Gerards and J. Kuper. Optimal dpm and dvfs for frame-based real-time systems. *TACO*, 9(4), 2013.

L. Gyarmati and T. A. Trinh. Power footprint of internet services. In *Proceedings of the 2Nd International Conference on Energy-Efficient Computing and Networking*, e-Energy '11. ACM, 2011.

D. He and W. Mueller. A heuristic energy-aware approach for hard real-time systems on multi-core platforms. In *Proceedings of the 2012 15th Euromicro Conference on Digital System Design*, DSD '12, pages 288–295, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4798-5.

P. G. Jansen, S. J. Mullender, P. J. Havinga, and H. Scholten. Lightweight edf scheduling with deadline inheritance. Technical report, University of Twente. May, 2003.

L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2), 1998.

C. M. Kamga, G. S. Tran, and L. Broto. Extended scheduler for efficient frequency scaling in virtualized systems. *SIGOPS Oper. Syst. Rev.*, 46(2), 2012.

M. Kerrisk. *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. No Starch Press, San Francisco, CA, USA, 1st edition, 2010. ISBN 1593272200, 9781593272203.

F. C. Klebaner. Introduction to stochastic calculus with application (3rd edition). 2012.

D. Lachut, S. Piel, L. Choudhury, Y. Xiong, S. Rollins, K. Moran, and N. Banerjee. Minimizing intrusiveness in home energy measurement. In *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, BuildSys '12, New York, NY, USA, 2012. ACM.

Y. M. Lee, L. An, F. Liu, R. Horesh, Y. T. Chae, R. Zhang, E. Meliksetian, P. Chowdhary, P. Nevill, and J. L. Snowdon. Building energy performance analytics on cloud as a service. *Serv. Sci.*, 2013.

X. Liu, P. Shenoy, and M. Corner. Chameleon: Application level power management with performance isolation. In *Proceedings of the 13th Annual ACM International Conference on Multimedia*, MULTIMEDIA '05, New York, NY, USA, 2005. ACM.

D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: Eliminating server idle power. *SIGARCH Comput. Archit. News*, 37, 2009.

A. P. Miettinen and J. K. Nurminen. Analysis of the energy consumption of javascript based mobile web applications. In *MOBILIGHT*, 2010.

S. Murugesan. Harnessing green it: Principles and practices. *IT Professional*, 10, 2008.

J. R. Norris. Markov chains. *Cambridge Series in Statistical and Probabilistic Mathematics*, 1998.

A. J. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma. Carat: Collaborative energy diagnosis for mobile devices. SenSys '13, New York, NY, USA, 2013. ACM.

B. B. Paleologo, L. Benini, A. Bogliolo, G. A. Paleologo, and G. D. Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18:813–833, 1998.

J. Park, S. Yoo, S. Lee, and C. Park. 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*, SEUS '09, pages 24–35, Berlin, Heidelberg, 2009. Springer-Verlag.

S. Patel and J. Perkinson. Fraunhofer report - the impact of internet browsers on computer energy consumption. 2013.

Q. Qiu and M. Pedram. Dynamic power management based on continuous-time markov decision processes. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, DAC '99, New York, NY, USA, 1999. ACM.

C. Reis and S. D. Gribble. Isolating web programs in modern browser architectures. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, 2009.

I. K. S. Rodrigues, R. Koren. A study on the use of performance counters to estimate power in microprocessors. In *Circuits and Systems II: Express Briefs, IEEE Transactions*, 2013.

E. Rotem, A. Naveh, M. Moffie, and A. Mendelson. Analysis of thermal monitor features of the intel pentium m processor. In *in Workshop on Temperatureaware Computer Systems*, 2004.

S. Russel and P. Norvig. Artificial intelligence: A modern approach (3rd edition). 2009.

L. Sharifi, N. Rameshan, F. Freitag, and L. Veiga. Energy efficiency dilemma: P2P-cloud vs. mega-datacenter (best-paper candidate). In *IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom 2014)*. IEEE, Dec. 2014.

H. F. Sheikh, H. Tan, I. Ahmad, S. Ranka, and P. Bv. Energy- and performance-aware scheduling of tasks on parallel and distributed systems. *J. Emerg. Technol. Comput. Syst.*, 8(4), 2012.

H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu. Achieving autonomous power management using reinforcement learning. *ACM Trans. Des. Autom. Electron. Syst.*, 18(2), 2013.

J. Simão and L. Veiga. Qoe-JVM: An adaptive and resource-aware java runtime for cloud computing. In *2nd International Symposium on Secure Virtual Infrastructures (DOA-SVI 2012), OTM Conferences 2012*. Springer, LNCS, Sept. 2012.

R. P. Singh, S. Keshav, and T. Brecht. A cloud-based consumer-centric architecture for energy data analytics. In *Proceedings of the Fourth International Conference on Future Energy Systems*, e-Energy '13, New York, NY, USA, 2013. ACM.

R. L. R. T. H. Cormen, C. Stein and C. E. Leiserson. *Introduction to Algorithms*. Higher Education, McGraw-Hill, 2001.

A. S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.

L. Veiga and P. Ferreira. Poliper: Policies for mobile and pervasive environments. In *3rd Workshop on Adaptive and Reflective Middleware (5th ACM International Middleware Conference)*, volume 6. ACM, Sept. 2004.

K. L. W. Liu and D. Pearson. Consumer-centric smart grid. *Innovative Smart Grid Technologies*, pages 1–6, 2011.

Y. Wang, Q. Xie, A. Ammari, and M. Pedram. Deriving a near-optimal power management policy using model-free reinforcement learning and bayesian classification. *Proceedings of the 48th Design Automation Conference on - DAC '11*, page 41, 2011.

M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. 1994.

Wiltzius, Tom, Kokkevis, Vangelis, and The Chrome Graphics team. Gpu accelerated compositing in chrome. http://www.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome, May 2014.

D. K. Y. Agarwal, R. Gupta and T. Weng. Buildingdepot: An extensible and distributed architecture for building data storage, access and sharing. In proc. of the 4th ACM Workshop on BuildSys. ACM, 2012.

H. Yan, D. K. Lowenthal, and K. Li. Ace: An active, client-directed method for reducing energy during web browsing. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '05, New York, NY, USA, 2005. ACM.

X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan, and M. E. Papka. Integrating dynamic pricing of electricity into energy aware scheduling for hpc systems. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13, New York, NY, USA, 2013. ACM.

S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto. Survey of scheduling techniques for addressing shared resources in multicore processors. *ACM Comput. Surv.*, 45, 2012.