POLITECNICO DI MILANO
Facoltà di Ingegneria dell'Informazione

POLO REGIONALE DI COMO

Master of Science in
Computer Engineering

Business Intelligence Methodologies applied to Green IT

Supervisor: Prof. Chiara Francalanci
Assistant Supervisor: Eugenio Capra
Giampaolo Agosta

Master Graduation Thesis by: Ricardo Planer
Student Id. Number: 736465

Academic Year 2009/10

POLITECNICO DI MILANO
Facoltà di Ingegneria dell'Informazione

POLO REGIONALE DI COMO

Corso di Laurea Specialistica in
Ingegneria Informatica

Business Intelligence Methodologies applied to Green IT

Relatore: **Prof. Chiara Francalanci**
Correlatore: **Eugenio Capra**
Giampaolo Agosta

Tesi di laurea di: Ricardo Planer
Matr.: 736465

Anno Accademico 2009/10

Agradecimento à minha família,
a vocês, que me deram a vida e ensinaram a vivê-la com dignidade, não bastaria um obrigado. A vocês, que iluminaram os caminhos obscuros com afeto e dedicação para que os trilhásse sem medo e cheio de esperança, não bastaria um muito obrigado. A vocês, que se doaram inteiros e renunciaram aos seus sonhos, para que, muitas vezes, pudésse realizar os meus. Pela longa espera e compreensão durante as longas viagens, não bastaria um muitíssimo obrigado. A vocês, pais por natureza, por opção e amor, não bastaria dizer, que não tenho palavras para agradecer tudo isso. Mas é o que nos acontece agora, quando procuro arduamente uma forma verbal de exprimir uma emoção ímpar. Uma emoção que jamais seria traduzida por palavras.
Muito obrigado!

SOMMARIO

Il consumo di energia relativo alla infrastruttura IT svolge un ruolo importante sui costi operativi, incide direttamente sul scalabilità del sistema e inoltre è importante per questioni di responsabilità ambientale. La ricerca dell' efficienza energetica IT ha sempre puntato sull' hardware, mentre nel dominio del software, quest' ultima è stata principalmente considerata per i sistemi embedded. In questo documento, sarà introdotto un quadro di un'architettura software che fornisce il supporto per migliorare l' efficienza energetica. L' accento sarà posto sul modulo Business Intelligence, che si occupa di analizzare i dati in uscita e presentare le metriche di valutazione.

Parole chiave: Business Intelligence; Green Software; efficienza energetica; qualità di software; sviluppo del software.

ABSTRACT

The energy consumption related to IT infrastructure plays a major role on the operational costs, in addition by affecting directly the system scalability but also by being important for environmental responsibility issues. Research on IT energy efficiency has always focused on hardware, while within the software domain it has mainly being considered for embedded systems. In this document, it will be introduced a software framework architecture that provides support to improve energy efficiency. The focus will be on the Business Intelligence module, which is responsible to analyze the output data and present the evaluated metrics.

Keywords: Business Intelligence; Green Software; energy efficiency; software quality; software development.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER

1. INTRODUCTION

In the very urge of fast information technologies development, much may have been lost regarding the quality of software systems. Although currently, new trends of research are focusing on Green IT, i.e. the study of the energy consumption of IT, is attracting both the academic and the industrial awareness. It has become an important subject for ethical, cost and scalability reasons (Murugesan, 2008).

First of all, IT infrastructures are responsible for 2% of the CO2 world emissions footprint, responsible for the greenhouse effect and consequently the first reason for global warming. Second, energy costs have dramatically increased and their impact on the overall IT infrastructural costs is becoming even more significant (e.g., according to Kumar, 2007, nowadays yearly power and cooling costs for servers are almost 60% of the initial purchasing cost). Moreover, energy requirements represent one of the data center scalability issues, since providers often have difficulties in supplying data centers with all the required energy (Lee and Brown, 2007). IT energy consumption sustainability is important from an economic, societal and environmental perspective for organizations. These three dimensions are overlapping factors for sustainability, but very often the economic and

societal are ultimately constrained by the environment. Energy efficient software can play an important role in these three overlapping spheres of sustainability.

Research has always focused on *hardware* energy efficiency, and only marginally on software. In particular, energy efficiency has been investigated mainly for embedded systems and low-level software (Sivasubramaniam at al., 2002; Fornaciari et al., 2001), and not at Information Systems (IS) level. Accordingly, hardware energy efficiency has significantly improved in the last years, with particularly high gains in the energy efficiency of mobile devices, as a response to battery autonomy issues. Over the past 30 years, the value of MIPS/W of mainframe systems has increased of a factor of 28.000 (ACEEE, 2008), which represents an improvement much higher than those achieved by production machines in other industrial sectors, such as steel production or automotive. The starting theoretical foundation of this work is that software is the main driver of power consumption on CPUs as it indirectly causes all the commutations performed by the processor and thus induces all the consumption of the infrastructural layers above (e.g., cooling, UPS, etc.) By analogy, in order to reduce car pollution it is important to increase the mileage per liter of gasoline, but also to optimize the trips in order to reduce the overall number of driven miles. Similarly, it is important to reduce the energy required by hardware to perform elementary computations, but also to optimize the number of computations required to satisfy a given set of functional requirements and workloads.

With respect to the power consumption of hardware components within a server, a CPU takes approximately 60% of the input power. Meanwhile, the power over memory and hard drivers are almost independently from its utilization once the power mostly goes to the RAM refresh cycles and disks spinning respectively. Thus, a better utilization of memory and disks could reduce the number of operations executed on the CPU, and consequently, minimizing the total amount of energy spent.

Summarizing, software can be more energetically efficient by means of:

- Optimization of the software code in order to reduce the CPU utilization;
- Usage of a less powerful CPU or by sharing the CPU with virtualization techniques;
- Improving the use of memory and disks, thus, reducing the CPU utilization;

Nevertheless, whereas hardware has been constantly improved to be energy efficient, *software* has not recorded a comparable track. The software development life cycle and related process management methodologies rarely consider this parameter. Not surprisingly, the over 50 ISO software quality parameters do not include energy efficiency [1] (cf. ISO 9126:2003). The prompt availability of increasingly efficient and cheaper hardware components has lead designers up to now to neglect the energy efficiency of end-user software, which remains largely unexplored. In the last decades research has focused on optimizing the energy consumption of operating systems, infrastructural component and embedded systems, for example by striving to develop power-efficient compilers (Daud, Ahmad and Murthy, 2009), but very little research has been made on the energy efficiency of end-user applications, and in particular of Information Systems (IS) software. A paramount difference between embedded systems and IS is that in IS contexts hardware and low level architectures are usually imposed and cannot be easily influenced. For example, from a low-level programming point of view an emerging technique for reducing energy consumption is the dynamic configuration of clock frequency (Huang, Li and Li, 2009), but if we assume an IS perspective it would be quite difficult for the CIO of a company that wants to improve the energy efficiency of the any software to use such technique.

This thesis is part of a wider project developed in collaboration with Politecnico di Milano [2] and Italian Financial Institutions, which intend to analyze the performance issues by exploring the concepts of Green IT, and later on to feedback better setup parameters to the system. Most of the work will focus on financial algorithms which are widely used on credit institutions and are rather computationally expensive. The document will focus on the

business Intelligence module, which is a JAVA application that can load the results from profiling and evaluated the performance of the framework as a whole. The document is structured as a brief introduction describing the scenario of *green* researches with motivations and objectives followed by some details regarding the Green Area and some important concepts. Then, it is presented the project itself with details about the analysis, development, execution and result interpretations.

CHAPTER

2. STATE OF ART

In this chapter is presented the current state of art regarding to research and development in the field of energy efficiency applied to information technologies. In the section 2.1, it is introduced the concepts of Green IT and the fundamental aspects are described. The section 2.2 introduces the concept of energy consumption from a software perspective and how to measure it. Finally, section 2.3 describes how software profiling is approached and why it is important in this work.

2.1 GREEN IT

Nowadays technology is an important part of our lives. The use of technology has advanced in several areas, improving life and work offering great advantages. Almost every appliance participates in technology and sometimes we do not even notice.

The expression Green IT is from now on commonly used to indicate a new field of research, focused on problems related to the energy consumption and environmental impact of information systems. Particularly, the term can be referred to two specific subjects:

- IT energetic efficiency;
- Management of ecological compliancy of IT lifecycle;

Both of them are directly related to control the usage of natural resources and their impact on the energy production chain.

More precisely, the first subject concerns to the improvement of energetic efficiency by means of improvements on infrastructure management and systems, but avoiding any undesired influence on legacy systems and the way they are used. We would say: a kind of external layer of optimization on an atomic system. It is important to be mentioned that in most recent years, the ratio between computational capacity and energy required has enhanced. This was mainly motivated by mobility and low-power devices.

The second subject looks upon the lifecycle of IT equipments development, energy needed in production, power consumption during the usage cycle and more.

2.1.1 IMPACT OF IT: ENVIRONMENT AND COSTS

In the last years, energy efficiency for IT is something which is assuming a growing importance, either by the problem of global pollution and warming, and either by the increasing costs sourced from infrastructural needs in order to keep an information system.

The Information and Communication system has a strong impact on carbon footprint and climate change. It has been estimated that in fact the IT infrastructure is responsible for 2% of global CO_2 emissions [3]. A research conducted by Gartner Group has estimated that a billion tones of CO_2 emissions are attributed by the IT sector, while the

total global amount is 49 billion of tones annually. As so, the reduction of energy consumption is the key factor to reduce the CO_2 footprint and its impact on global warming.

The IT interferes in the environment in different ways. The whole computer lifecycle, from its manufacturing to disposal, is accountable for several environmental problems. The chain of production of a computer consumes electricity, chemicals and water which cause pollution. To illustrate the distribution of energy during a computer lifecycle at the early 21st century, approximately 72% of whole energy used by a Personal Computer (PC) was spent in its production while the others 24% consumed in its 4 years mean usage. While now, the statistics have inverted, once 23% of whole energy is consumed by a PC in its production and 72% used during 4 years of usage. Then, one can infer that production process has been optimized, reducing the amount of energy required, while the energy demanded by usage has increased.

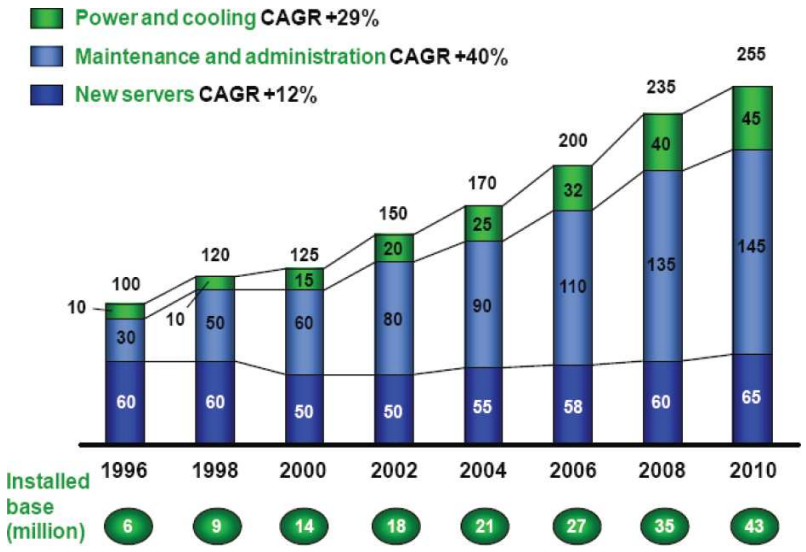
4 Years Lifecycle					
Source	**Production**	**Distribution**	**Usage**	**Disposal**	**Total**
[4]	6060 MJ **(72%)**	336 MJ **(4%)**	2020 MJ **(24%)**	-	8416 MJ
[5]	5801 MJ **(76%)**	-	1756 MJ **(23%)**	<76 MJ **(<1%)**	7633 MJ
[6]	3709 MJ **(26%)**	713 MJ **(5%)**	10270 MJ **(72%)**	<142 MJ **(<1%)**	14264 MJ

TABLE 2.1 - ENERGY CONSUMPTION DURING PC LIFECYCLE

Focusing on the energy required in order to run a server (computer, monitor, network and refrigeration) in a data center is constantly increasing. In fact, every personal computer (PC) produces a ton of CO_2 annually [13, 12], while a server manufacturing requires the same amount of CO_2 emitted by a *sport utility vehicle* (SUV) to travel 25 Km. The intensified technological development supports the creation of new processor s, every time faster and more power consuming. A common Pentium IV dissipates 120W, an increase of 120% with respect to an average 486 that consumed 10W.If one considers that

a modern *blade server* consumes 1kW, as so as a refrigerator; a rack would consume 40kW, an entire building. An average data center consumes 250kW, as a neighborhood, while big data centers reach 1MW, an entire city.

In the last fourteen years, as one can see at figure 2.1 and 2.2, the cost of new hardware has grown less than the cost of power and cooling:

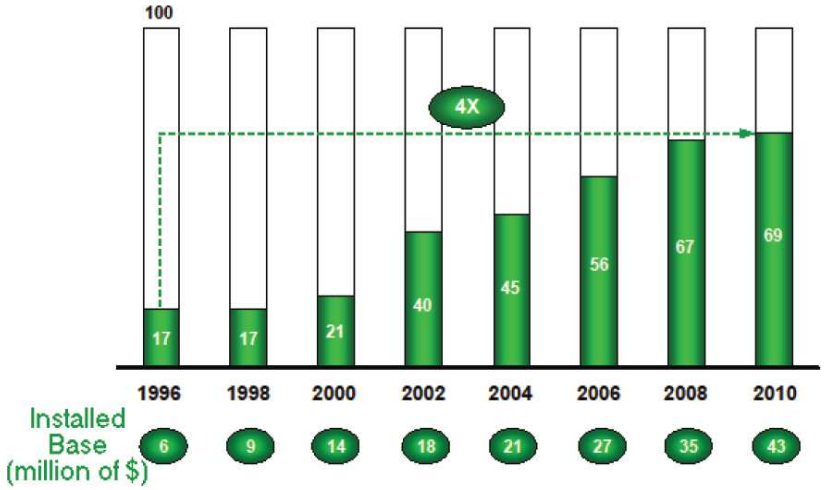


Source: IDC (2006)

FIGURE 2.1 – GROWING RATE OF IT INFRASTRUCTURE COSTS.

* CAGR - Compound annual growth rate

IDC estimates [7] that for each $1.00 spent on a new server, $0.70 cents are spent on power and cooling. The data shows that costs on power and cooling increased four times, also considering the increase on energy rates.



Source: IDC (2006)

FIGURE 2.2 - RATIO BETWEEN COSTS OF NEW SERVERS VS POWER/COOLING

It is not only the cost of energy that is continually rising; but also the consumption rises constantly (8-10% yearly). This phenomenon is limiting the scalability of big data centers at high density urban areas. According to Forrester Research, in the next years 60% of data centers are going to be limited by energy consumption, space and cooling [8].

2.1.2 SUSTAINABLE IT: GREEN IT

Green IT or Green Computing is the study and application to design, production, use and disposal of a computer, server and associated sub systems, in an efficient way and with the least environmental impact. Green IT seeks to meet the economic interest and performance improvement considering ethical and social responsibility that derives from the recent precarious ecological conditions.

The first occurrence of Green IT was in 1992, initiated by U.S. Agency responsible by a program created to promote energy efficiency in electronic equipments. From this

program, some new resources started being adopted in systems design such as Sleeping and Stand By mechanisms to save energy.

Also, a certification program created by a Swedish organization called TCO started evaluating equipments related to its magnetic and electrical emissions like Cathode Ray Tube display and later every kind of energy consumption components. In 2006 the certification program created in 1992 established a strict layered ranking for all computers components and consequently the products which use such electronic parts.

There are three major areas where Green Computing is concerned:

- IT energy efficiency;
- Eco-compatible management of IT lifecycle;
- IT as a instrument for a *green* governance;

The IT energy efficiency can be improved by acting both on the design and management of structures and data centers, and by changing the corporate culture and practices of use. Actually, the energy efficiency (the relationship between energy consumption and performance) has grown in the last years once performance has been enhanced while energy consumption stayed steady. Considering the benchmark TPC-C [9] widely used to assess the processor performance, the energy efficiency can be measured as million transactions minute per absorbed Watt (Ktpm-c/Watt). The value of this index has improved by a factor of 2.5 on the last decade. As the demand of computational capacity grows as so the energy consumption, improvements in this area is essential.

The eco-compatible management of IT lifecycle includes all phases, from production to disposal. Pollution caused by IT not is merely attributable to the consumption of electricity but derives, in part, from the incorrect use and disposal of toxic substances used during fabrication. The *Waste of Electric and Electronic Equipment* (WEEE) is defined as a directive at the European Union (2002/95/CE) that establishes precise rules for recycling

and recovery of such waste [10]. As a matter of fact, pollution derived from IT is responsible for 70% of soil-derived contamination by plumb, cadmium and mercury

The IT as a instrument for a *green* governance, by detecting and measuring relevant parameters throughout the whole business processes. Recent studies [3] have shown that 86% of ICT departments in United Kingdom have no knowledge of their weight of CO_2 emissions, while 80% of organizations have no awareness of electricity expenditures. In order to have a more sustainable IT is necessary a deep change of culture and corporate management. The impact has to be monitored by appropriate KPIs (Key performance Indicator) along with the other usual indicators.

2.1.3 Green IT: Solutions

In the next paragraphs, some examples of important approaches applied to Green IT, in order to solve real issues by means of auxiliary software, hardware improvement or both.

As a software method to increase the resource utilization and sharing, there are virtualization techniques, which achieve power savings at system level. Virtualization is a creation of virtual resources over a single hardware. It is usually created using modified Operational Systems running into management software in attempt to reach the maximum usage of resources, replacing the necessity of several physical computers under loaded. The guest software or operational system runs as if it was installed normally into a single physical platform. Widely used nowadays, although it requires a good knowledge in performance assessments and resources configuration. Virtualization, therefore, offers various advantages. A virtual machine can be more easily controlled and inspected from outside than a physical one. New virtual machine can be created without the need to purchase more equipment. Virtual machines can be easily migrated to different physical

platforms once they are ultimately just files. Hence, virtualization is very important in Green IT since it allows substituting several physical systems by virtual machines hosted in just one single powerful system, thereby unplugging the original hardware and reducing powering and mainly the cooling consumption.

Terminal Servers in last years have also been used to put the computing operations in only the central computer, while terminals connected to it are simply thin clients and text terminals with low power consumption. This solution offers advantages in security and availability, since with a terminal breaking the service can still be provided. Advantages can be found out in terms of energy optimization. In fact, thin clients use no more than 10% of a regular workstation.

The Power Management for computer, also known as Advanced Configuration and Power Interface (ACPI), is an open standard which defines protocols to allow the operational system to control the power state of underlying hardware. The OS can turn off autonomously a set of peripherals such as monitors and hard drivers during inactive periods. The hibernation and stand by modes do use this standard. There are also programs to give users the possibility to configure the CPU voltages where the user will reduce of increase powering, frequencies and also heat dissipation. Modern CPUs are able to configure its frequency autonomously according to the current load.

Power Suppliers are usually 70% efficient with 30% of energy lost in heat. This means that for each 70W generated the supplier requires 100W in input. Better quality power supplies can be over 80% efficient. Thus there is less energy wasted in heat and consequently less power is used in cooling. The industry initiated the 80 PLUS which is a initiative to manufacturing of powering units with more energy efficiency. It certifies products using some different parameters of load and power factor. From July 2007 all certified desktop computers are guaranteed to work with a maximum of energy waste of 20% of all powering capacity.

Other important topic is the displays, where sensitive enhancements were achieved in last few years. Displays have slashed their power usage from an average value of 110W to 40W in last decade representing a reduction of more than 60%. It was feasible since a migration from CRT monitors to LCD occurred.

2.2 GREEN SOFTWARE

2.2.1 FROM HARDWARE TO SOFTWARE

The tiniest quantity of information is represented by a bit, which could be physically associated to the presence of a charge or the state of a magnetic field. In CMOS technology, which is used in the modern processors, the power consumption of the simplest structure is divided in:

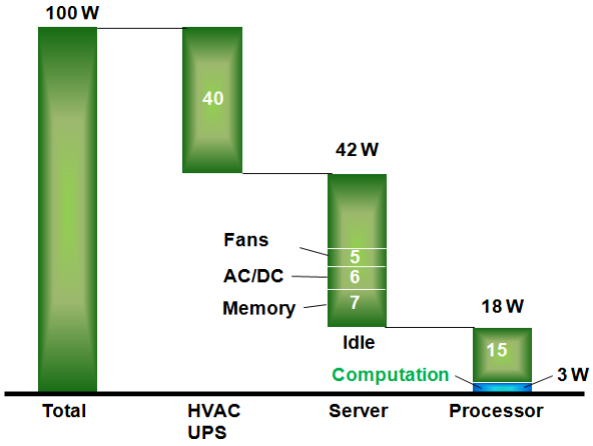
- **Static**: This refers to the energy spent to keep a CMOS circuit working (i.e. sub-threshold leakage, Diode/Drain leakage and Gate Leakage). They depend on physical characteristics of the technology only.
- **Dynamic**: Dynamic power is the power spent which depends on the activity inside the circuit and proportionally dependent of frequency. Transition power is the energy spent to charge the output capacitor on transitions from 0 →1. Short-circuit power depends on the amount of time the circuit is short-circuited, occurring in both transitions 0→1 and 1→0:

$$P_{transition} = C_L * V_{cc} * \alpha_{0\to1} * f \quad (2.1)$$

$$P_{short-circuit} = t_{sc} * V_{cc} * I_{peak} * P_{0/1\to1/0} * f \quad (2.2)$$

A study conducted at the Massachusetts Institute of Technology (MIT) has fixed the lower bound to the energy required to commute one bit between states at a certain frequency. This limit is defined when every bit is associated to a quantum electron spin. A computer using quantum electronic could achieve the 10^{-25}J to commute a single bit at 1GHz, while in a traditional computer is 10^{-16}J. This difference is due to the fact that commutations occur in the lowermost in a computer system: all the upper levels contribute to multiply it by a factor of 30.

In figure 2.3 it is shown the subdivisions regarding to the energy consumption in typical a data center. It is clear how the energy efficiency problems have to be treated in each level of the structure. Also, it is important to notice that to improve the lower levels is fundamental. For instance, when the processor runs a software that is inefficient, and which ultimately drives the amount of commutations, it will increase the amount of power needed in order to execute. It starts a chain reaction that is seen in the upper levels, for example, requiring the cooling system to act. Nowadays, the energy efficiency of IT systems are estimated in theory to be 50% maximum.



Source: IBM (2007)

FIGURE 2.3 – PERCENTUAL ENERGY CONSUMPTION OF A DATA CENTER

This work will focus in the software layer related to large Data Center, for example, from Financial Institutions. Must be known that even the Figure above showing that the software may handle a little amount of power consumed comparing to every part in architecture, the software has a dramatic interference in the final value because it defines the relation between idle and busy states of processor, and also can measure the performance requirements. These requirements are able to address the resources which must be available to the system, and also address possible virtualization.

Even though software does not consume energy directly, it drives the energy usage in all levels of the system. From the physical point of view, the mean power consumption of a processor running any application is:

$$P = V_{cc} * I \tag{2.3}$$

Variable P is the power, I is the mean current used and Vcc the powering voltage. Then, the energy consumed by the processor running software is the integral of Power in function of time, verifying:

$$EC_{physical} = \int V_{cc} * I \, dt \tag{2.4}$$

Therefore, to calculate the energy consumed by the system it is required to measure the current and voltage powering the system. Moreover, the measures are platform dependent and it is obvious that only the measure of these two components does not guarantee the knowledge of each consumption points of the system. To analyze how an application demands a certain amount of energy to compute, it is necessary to link Physical and Logical domains

From the logical perspective can be inferred from Margolous Levitin theorem [11] that the maximum frequency in state commutation from a physical system is directly

proportional to the whole energy of the system. So, the minimum amount of commutation energy needed by a system to work properly in a certain frequency is calculated with:

$$E_{min}(f) = \frac{f * h}{4} \quad (2.5)$$

In the previous formula, *f* is the frequency and *h* is Plank constant. For example, one bit with the electron spin direction, the energy required to commute at a 1GHz frequency is $E_e \cong 5\,.\,10^{-21}J$.

Going deeper, we reach other variables important in energy measuring. One of them is the systems information measure used to define the amount of information needed by it (i.e. number of bits). Also, we have entropy which is a level of disorder of data which can be transcript as the computational complexity (Cc) for a desired output to be generated by a system and the Thermodynamic deepness (Td) which is defined as the number of bits neg-entropic required in order to build a system. These two last components respectively Cc and Td, generate the logical Energy Consumed formula:

$$EC_{logical} = E(f) * C_c * T_d \quad (2.6)$$

Finally, it is possible to design a methodology that allows one to compare different applications from an energetic point of view. From expressions 2.4 and 2.6, one can infer that energy consumption can be an approximation of energy from a logical perspective:

$$EC_{physical}\ \alpha\ EC_{logical} \quad (2.7)$$

2.3 SOFTWARE PROFILING

Profiling the software is a resource available which allows the stakeholder to verify where a program spends its time and which functions are being called while the whole system is executed. This information can show which pieces of the program are slower than is expected and those that might be candidates for rewriting or substitution to make the program execute faster and save power and time. Moreover, other output is the presentation of how often certain functions and procedures are being called for execution. This may help to spot code blocks interesting for deep analysis.

Since the profiler uses information collected during the actual execution of your program, it can be used on programs that are very large and/or complex to analyze by reading the source directly. However, the way the program is running will affect the information that shows up in the profile data. Thus, is mandatory that all the important features and those with high performance influence are stimulated to provide reliability and usability of profile information generated in profiling analysis. Software profiling has some generic steps:

- Must be enabled profiling and/or the program must be compiled/linked with profiling support;
- The program must be executed in a desired way to generate relevant profile data;
- Must be executed a tool to interpret the generated data.

Depending on the language or platform (i.e. Java, C, C#) the steps above might be joined or transparent for the user. Profilers use a wide variety of techniques to collect data, including hardware interrupts, code instrumentation, instruction set simulation, operating system hooks, and performance counters. Summary profile information is often shown against the source code statements where the events occur, so the size of measurement data is linear to the code size of the program. In contrast, the size of a (full) trace is linear to

the program instruction path length, making it somewhat impractical. For sequential programs, a profile is usually enough, but performance problems in parallel programs (waiting for messages or synchronization issues) often depend on the time relationship of events, thus requiring a full trace to get an understanding of what is happening. Our work is based mostly on sequential programs, focused in mathematical/financial intensive execution methods.

CHAPTER

3. OVERALL METHODOLOGY AND GREEN IT ARCHITECTURE

This chapter will describe the methodologies that supported the creating of this thesis, from the definition of the problem to the solution used in order to construct the final architecture. Also, some of the implementation details are discussed.

3.1 GENERAL OVERVIEW OF THE PROBLEM

There are many hypotheses regarding the energy efficiency of an IT system. The general idea is that the energy saving derives from the data center structure optimization. As for every Watt spent to supply a server CPU, 28 Watts are required to the hosting data center. As so, the main idea was that to improve the energy efficiency would be necessary to optimize the infrastructure only, and that software has no impact on consumption,

Finally, we tend to commonly associate the energy efficiency to performance; and that improving the performance of the software is related to the reduction of CPU usage

and, consequently, to energy savings. Nowadays, after industrialization of software, performance or energy saving is not primarily concern, but with maintainable software. This is so because the most representative costs are related to software maintenance, therefore, costs are reduced when maintainable software is developed. Unfortunately, the ISO regarding software quality does not consider energy efficiency yet.

3.1.1 THESIS IMPACT OVER CORPORATE SOFTWARE

As every corporation should start thinking of Green IT and to define its own strategies to address the problem. This work could demonstrate a possible approach towards the progress of green computing targeted to companies that does not want to modify radically its infrastructure.

This thesis proposes an architecture that is tactically incremental, which the objective is to keep the infrastructure and business processes intact. The idea is to insert code modification at some key points where software could be optimized by the use of data tabulation. Therefore, the only thing that will be modified are some specific portion of the software code, although, not changing the expected behavior nor results.

3.1.2 TABULATION AS METHOD FOR SOFTWARE OPTIMIZATION

The use of pre-calculation and tabulation is already been used widely by the cryptography field where computational expensive algorithms are common, tabulation is used in order to improve efficiency and make it possible to break cipher codes in reasonable time. Usually an attack to cryptographic systems requires exhaustive algorithms and therefore massive computing power and time. When the same attack has to be reproduced

more than once, the attacking system can pre-calculate all the necessary values and store it in memory. The attack then is “instantaneous”.

Tabulation can contribute to improve the performance of software and thus its energy consumption. The underlying idea is that instead of demanding the CPU to compute the values (in the CPU, energy is a function of load); the data is retrieved from memory in its place (in the memory, energy is constant and independent from use or load).

In order to choose consciously what prerequisites are important in order to decide whether or not a portion of code is suitable for tabulation, it is important to analyze the behavior aspects of the modified function against the normal execution flow. In the figure 3.1, we identify the possible change in the flow at the modified piece of code, from the original execution flow on the left:

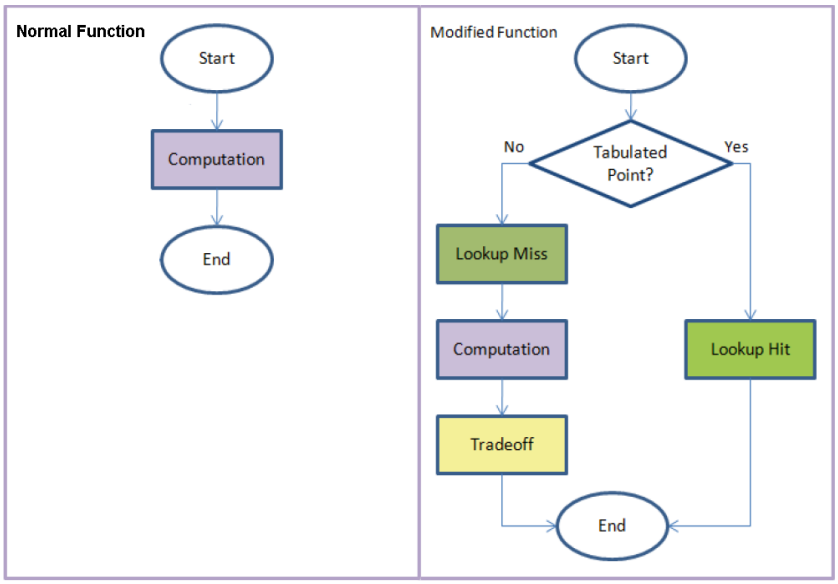


FIGURE 3.1 – ILLUSTRATION OF THE SYSTEM BEHAVIOUR BEFORE AND AFTER MODIFICATION

As one can see, the introduction of the mechanism for tabulation also inserts overhead when the original code has to be executed. From a time perspective, it is possible to identify the following timing information:

- **Lookup Miss Time (T_{miss})**: Time spent to look for a specific value that is not tabulated;
- **Computation Time ($T_{computation}$)**: Time spent to compute the original code;
- **TradeOff Time ($T_{tradeoff}$)**: Time spent by the trade off module to decide either to tabulate or not the value;
- **Lookup Hit Time (T_{hit})**: Time spent to look for a specific tabulated value;

Moreover, to better describe the model behavior, we extend it by considering also the following variables:

- **Hits Number (N_{hits})**: Number of times a point was found tabulated;
- **Miss Number (N_{miss})**: Number of times a point was not found tabulated;
- **Hits Percentage (α)**: Percentage of hits provided by the system;

$$\alpha = \frac{N_{hits}}{N_{hits}+N_{miss}} \text{ , where } 0 \leq \alpha \leq 1 \quad (3.1)$$

- **Modified Computation Time ($T_{modified}$)**: Time spent to compute the modified function;

$$T_{modified} = \alpha * T_{hit} + (1-\alpha) * (T_{miss} + T_{Computaion} + T_{tradeoff}) \quad (3.2)$$

- **Time saved ($\triangle T_{saved}$)**: Difference between Computation Time and Modified Computation Time;

$$\Delta T_{saved} = T_{computation} - T_{modified} \quad (3.3)$$

- **Break-even Hits Percentage (α_{be})**: Percentage of hits needed to achieve the break-even point, that is, the point where the timing of the original code is the same with tabulation mechanism;

$$\Delta T_{saved} = 0 \rightarrow T_{computation} = T_{modified}, \text{ substituting 3.2}$$

$$T_{computation} = \alpha * T_{hit} + (1 - \alpha) * (T_{miss} + T_{Computaion} + T_{tradeoff})$$

$$\alpha_{be} = \frac{T_{miss} + T_{tradeoff}}{T_{miss} + T_{computation} + T_{tradeoff} - T_{hit}} \quad (3.4)$$

Bearing that in mind, α can be used to distinguish between the energy saving potential for a given function. Furthermore, the best functions to tabulate are the ones with $\alpha_{be} \rightarrow 0$ once it is also the minimum value of α that the system needs to provide in order to save execution time. Moreover, to give an idea of alpha behavior, we can visualize it in the following chart, considering that the only variable value is the $T_{computation}$ on the horizontal axis:

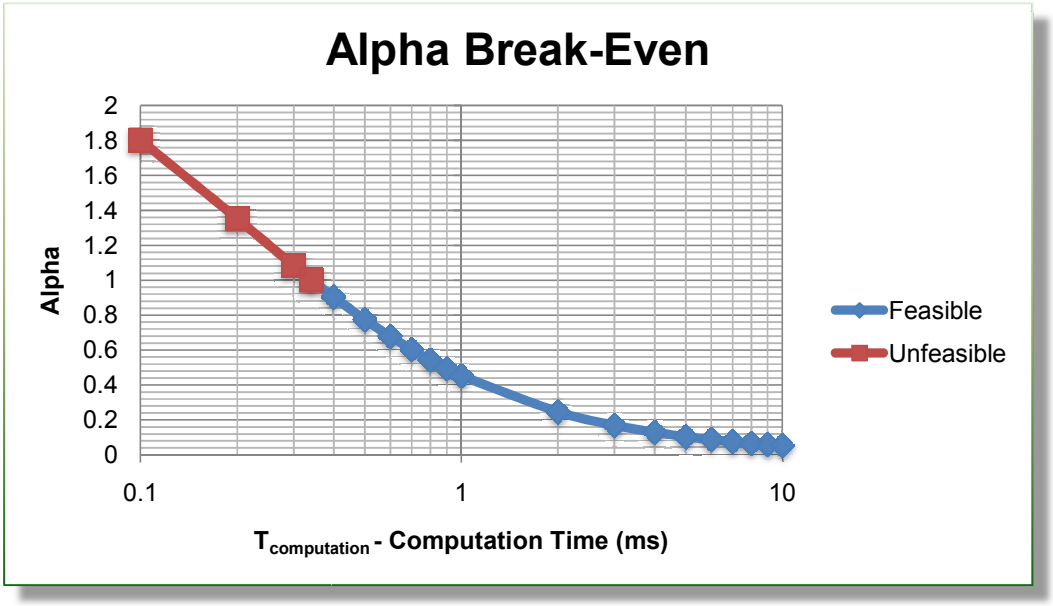


FIGURE 3.2 - ALPHA VALUE AGAINST COMPUTATION TIME

From the chart, we identify that there are values of $T_{Computation}$ that are unfeasible ($\alpha > 1$) due to the overhead inserted by the mechanism, and therefore, would not be advantageous to tabulate.

Considering that only *pure functions* with computation intensive calculations are going to be tabulated, which means, the processor utilization during the computation of the function is 100%, then in this case one can derive the energy consumption of a single modified portion of code during active execution as being:

$$E_{normal} = (P_{memory} + P_{cpu\ active}) * T_{computation} \quad (3.5)$$

Considering that $P_{cpu\ active}$ is the power spent by the processor during full utilization, we could rewrite it as being function of $P_{cpu\ idle}$ as so:

$$P_{cpu\ active} = P_{cpu\ idle} + \Delta P_{cpu}\text{ , where } \Delta P_{cpu} = P_{active} - P_{idle}$$

$$E_{normal} = (P_{memory} + P_{cpu\ idle} + \Delta P_{cpu}) * T_{computation} \quad (3.6)$$

In the other hand, when data is tabulated the amount of energy changes and depends on the flow of the tabulation mechanism, which is the average time of execution of the modified portion of code considering the time needed to seek the value in memory (lookup time) and whether or not it is already present in the table. Also, we can consider the utilization β of the processor during lookup. We can define that energy goes accordingly to the following equation:

$$E_{tabulated} = (P_{memory} + P_{cpu\ idle} + \beta * \Delta P_{cpu}) * T_{modified}\text{, where } 0 < \beta \leq 1 \quad (3.7)$$

Ultimately, from the difference of energy spent on normal and tabulated execution, we can compute the break-even point considering power:

$$\Delta E_{saved} = E_{normal} - E_{tabulated} = 0 \rightarrow E_{normal} = E_{tabulated},$$

substituting 3.6 and 3.7

$$(P_{memory} + P_{cpu\ idle} + \Delta P_{cpu}) * T_{computation} = (P_{memory} + P_{cpu\ idle} + \beta * \Delta P_{cpu}) * T_{modified}$$

as $T_{computation} = T_{modified}$

$$T_{computation} = \beta * T_{modified}$$

$$T_{computation} = \beta * (\alpha * T_{hit} + (1 - \alpha) * (T_{miss} + T_{Computaion} + T_{tradeoff}))$$

$$\alpha_{be} = \frac{\beta * (T_{miss} + T_{computation} + T_{tradeoff}) - T_{computation}}{\beta * (T_{miss} + T_{computation} + T_{tradeoff} - T_{hit})} \quad (3.8)$$

Note: if the CPU utilization β is 100% the break-even point is the same as before.

3.1.3 PURE FUNCTION DEFINITION

The definition of *pure function* can be derived from the concept of *mathematical function*, which is, a one to one mapping between input and output [14]. An software application method can be considered pure if it satisfies the following properties:

- Does not produce any side-effect, that is, the method does not produce any visible effect else than the result;
- Is Deterministic, that is, the method behavior depends only from the input parameters;

In more detail, a function is free of side effects if the only objects the method modifies are created within its execution. This definition let the method to create and modify new objects but without allowing it to change variables that are observable outside the method scope. In addition, a method is not allowed to cause any side effect outside the system environment (i.e.: write in a file, print messages to the console and communicate on the network).

```
public class ClassA {
 public ClassA() { }

 private int state = 0;
 public int Value(int a, int b) {
  int r = a + b;
  r += a * b;
  r += a - b;
  r += a/b;

  return r;
 }
}
```

```
public class ClassB {
 public ClassB() { }

 private int state = 0;
 public int Value(int a, int b) {
  int r = a + b + state;
  r += a * b;
  r += a - b;
  r += a / b;
  state = r;

  return r;
 }
}
```

At the previous example, the function *Calculate* from *ClassA* would have no problem to be tabulated once it does not alter any member of the class. Although, in *ClassB* the method stores the result into the member *state*, which is reused in every call; if this function is tabulated, this member would not be updated.

Moreover, a mathematical relation is considered function if every input is associated to an specific output; for two assessments of a mathematical functions with the same set of inputs, it will generate the same results. The deterministic requirement of a method is analogous to the mathematical function. The results must depend only on the arguments passed as input parameters besides any global state or variable (i.e.: time of the day).

By this property, one can define the concept of equality between parameters applied to a programming language. In the case of primitive types, the equality can be assessed by simply comparing the actual values of the variable. Although, regarding complex classes, it is necessary to question: Can tow different call to a method be equivalent if the inputs are the same but with a different alias? If possessing the same alias but residing in different memory addresses? There is not definitive answer to the previous questions because to any problem we have to deal will be an optimal choice rather than the other ones. Therefore, the functions determinism is a parametric property given the

definition of equality between parameters; a method is deterministic if for every call with equivalent arguments produces a undistinguishable result.

3.2 GREEN IT ARCHITECTURE

The architecture is developed to run at a JAVA environment and it is composed by three main components: the static analysis of the application *bytecode* to identify pure functions present on it; the other component is responsible to execute the *Decision Maker* – a meta-model that represents the portion of modified code; a *business intelligence* component that reports the performance of the system and is the object of this thesis.

The static analysis searches the whole application *bytecode* for pure functions. The collected information (method name, signature, number of *bytecode* instructions) regarding every function is then used by the *bytecode Modification* component to build the meta-module *Decision Maker*. Indeed, before any modification it is necessary to verify if what functions are suitable to be tabulated by using the *alpha* approach and some profiling information.

During the executing of the modified application, the Java Virtual Machine (JVM) executes it normally. Though, when it reaches a pure function, the JVM will not load the original code from the class where the function was, but will load the modified instead. The method, before the normal execution flow, performs a lookup into the table corresponding to the current function passing the parameters that were used: if there is a hit, which means, there is a return value already tabulated for those input parameters that were inserted in a previous execution, then the flow returns the value found; otherwise the function is then called and the set {Input Parameters → Return Value} is passed to the Trade Off module.

This component decides whether or not to save the result set to the table by following the internal logic of memory optimization and energy saving.

Finally, the B*usiness Intelligence* component is the one responsible to provide a report of the data related the architecture execution and behavior (executions time, functions frequency, memory utilization, and energy savings) by means of tables and charts. The main reason is to support validation of the solutions developed against energy efficiency and to suggest new parameters that could improve even more the results.

To clarify how components are connected and communicates with other modules, the following diagram shows the high level modules introduced before:

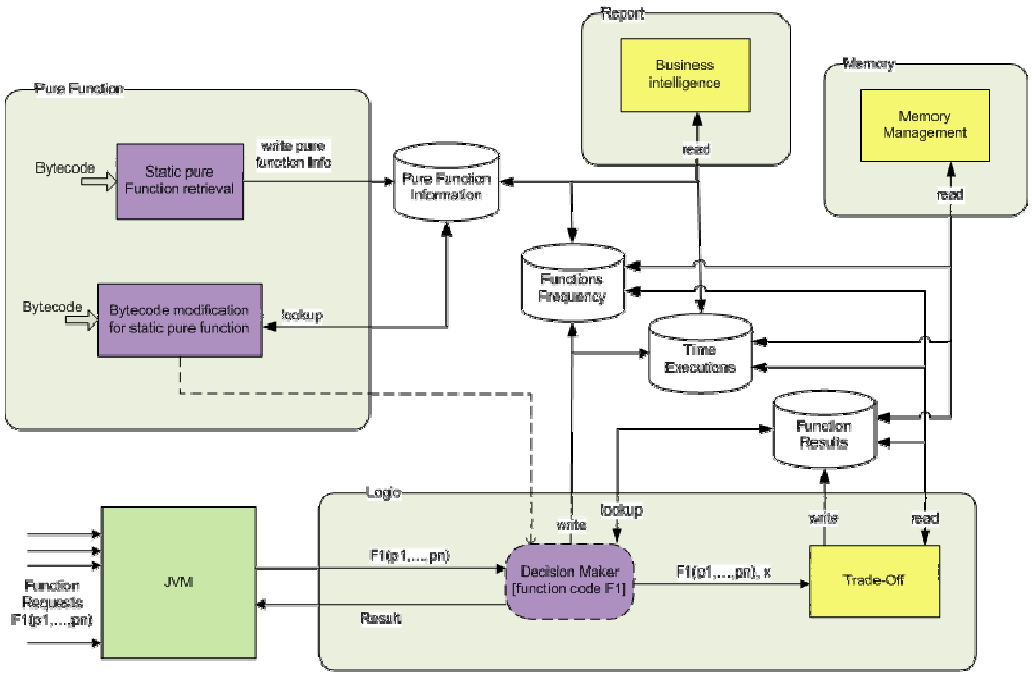


FIGURE 3.3 - HIGH LEVEL ABSTRACTION OF THE ARCHITECTURE

3.1.1 IMPLEMENTATION

The next step is to implement the architecture and the mechanism that enables the system to perform. In the following paragraphs it is described the main elements that enables the system to work.

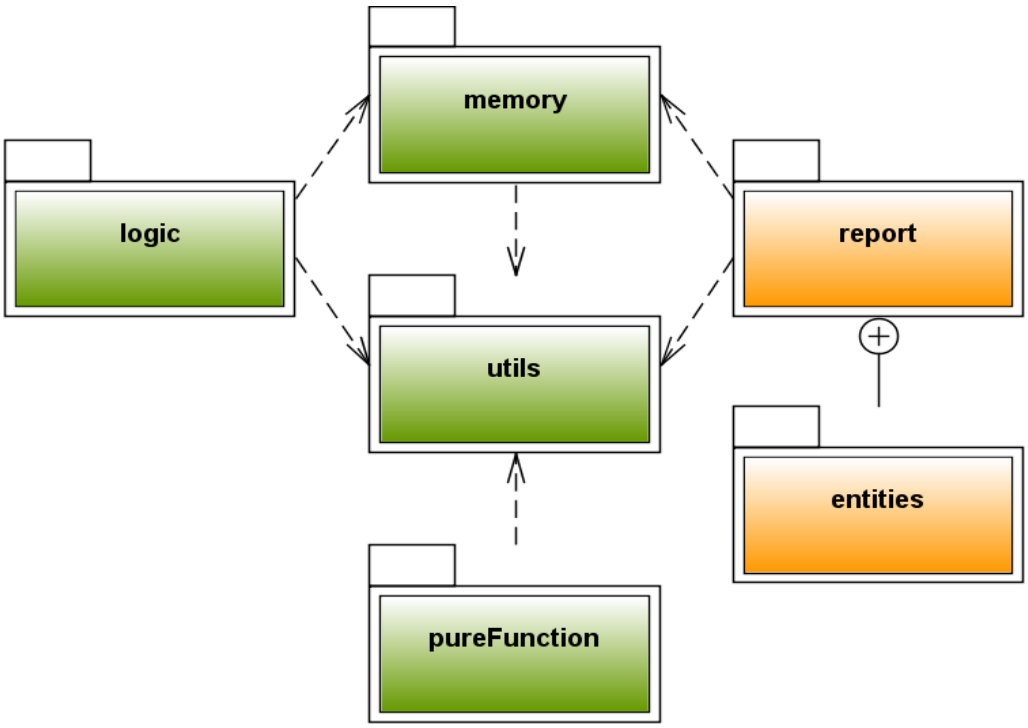


FIGURE 3.4 - PACKAGE DIAGRAM OF THE ARCHITECTURE

To better separate all the sections in a logical structure, the application code is organized in six different packages:

- **logic**: contain the classes responsible to evaluate if a call is eligible to be tabulated, given information on available memory, functions priority and the alpha values;
- **memory**: classes in this package control the functionalities related to access to the stored values and assessments of memory utilization;

- **utils**: contain supportive classes common to the other packages, i.e.: function, parameters and helper classes;
- **pureFunction**: collection of classes used to execute the static analysis of the bytecode and evaluation of eligible methods as *pure functions;*
- **report**: package containg the business intelligence classes, which are used to evaluate the results of the architecture performance;
    - **entities**: analogous to the utils package, containing particular classes to be used by the business intelligence only;

LOGIC PACKAGE

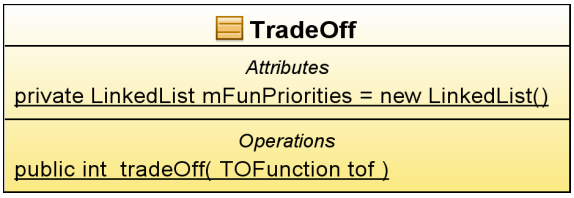


FIGURE 3.5 - CLASS DIAGRAM OF LOGIC PACKAGE

The most important action - operation available to the actor (Decision Maker: a conceptual module present in the whole project architecture) – is the TradeOff. The action consists of an intelligent algorithm which determines whether a p*ure function* and its entries (parameters and return) will be kept in memory, characterized as a Cache Memory, to improve the relation between computing and power consumption. Since the operations will be reduced by the storage of determined values needed by the whole system, the consumption of the processor and related electronic components will reduce.

The TradeOff uses some operations provided by the Memory Management like insertions, updates and deletions: InsertData, UpdatePriorities and RemoveData, respectively. After verifying the logic and deciding the including or not of a function into the

memory, it's responsible to operate the memory sending requests to handle data. These requests will keep also the memory state (Full, Available) and will provide methods to remove functions with lower priorities to save new data.

MEMORY PACKAGE

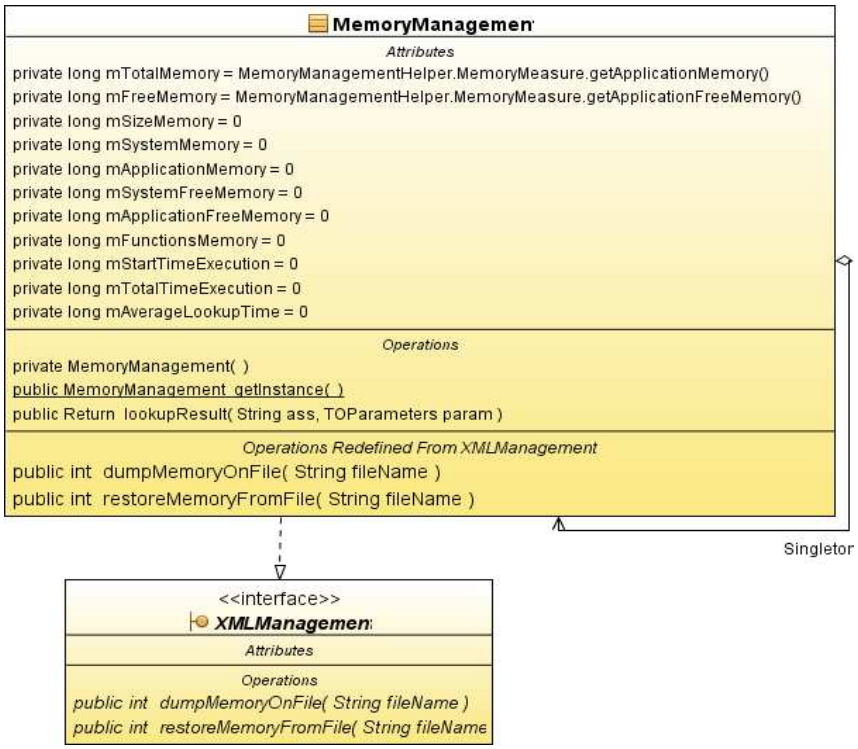


FIGURE 3.6 - CLASS DIAGRAM OF MEMORY PACKAGE

The *MemoryManagement* class is by definition a singleton – a design pattern that restricts the existence of objects of the class to one instance. It also provides functionalities to dump and restore its data to a file (Vide appendix B). Although, the most important function in *MemoryManagement* class is Lookup Result. It's a simple and easy set of steps to consult a required data in the memory. The function receives a function structure with parameters - from the Decision Maker - and is responsible to follow the *HashMaps* and

return a miss if the data is not present in the cache and the system will need to execute the whole mathematical method to get the result or the result itself.

UTILS PACKAGE

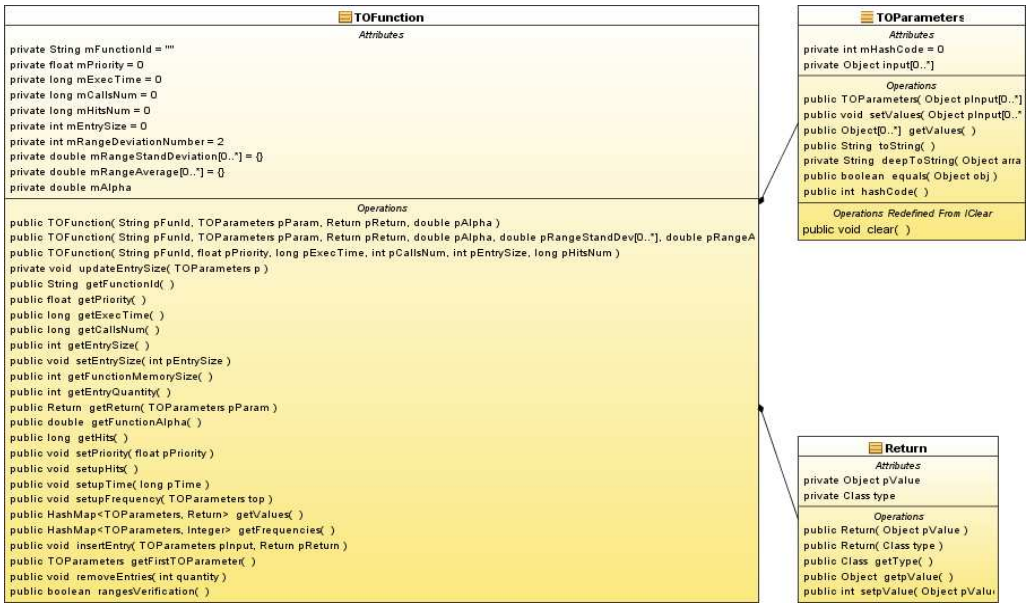


FIGURE 3.7 - CLASS DIAGRAM OF UTILS PACKAGE

The *TOFunction* class is the representation of a pure function inside the *MemoryManagement*. It collects a series of metrics regarding function calls to the real function, such as: priority, execution time (ns), number of hits, number of calls and so on. Each *TOFunction* contains a collection of all the parameters once passed as argument at a call to the function and is defined as a *HashMap<TOParameters. Return>*.

The other classes *TOParameters* and *Return* represent the actual data values passed as the function input and the return value respectively. The class *TOParameters* is projected to support only primitive types (i.e.: int, double, long, string, etc…) and arrays of primitive types of one level deepness.

REPORT PACKAGE

This package is the focus of this work and is full detailed at chapter 4.

LOOKUP EXECUTION FLOW

These sequence diagrams were built in order to provide an idea of internal module interaction. The Decision Maker module is the process trigger, starting with a table lookup it will use the method provided by the *MemoryManagement* to obtain any point needed by the system. The input is the function parameters, which contains all data used by the module to recognize the function and the parameters.

After asking a specific point, the *MemoryManagement* starts fitting the pure function data into its own objects. The first step is to generate a function key that will be used to distinguish different functions and types required. After having the Key it is possible to identify the Function but not a specific point inside the values table. So, the role of the *TOParameters* is to hold the parameters objects with the purpose to tell apart between all the points on the value table. At this point of the sequence all the information needed is already translated and the *MemoryManagement* will check if there exists the value for this parameters. If yes (Hit) the value is returned and the sequence is over as seen in figure 3.8:

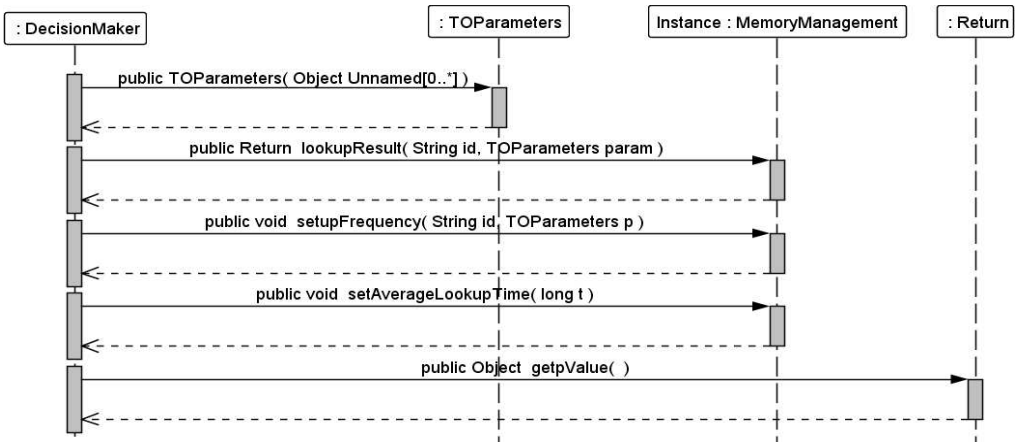


FIGURE 3.8 - LOOKUP EXECUTION FLOW – HIT

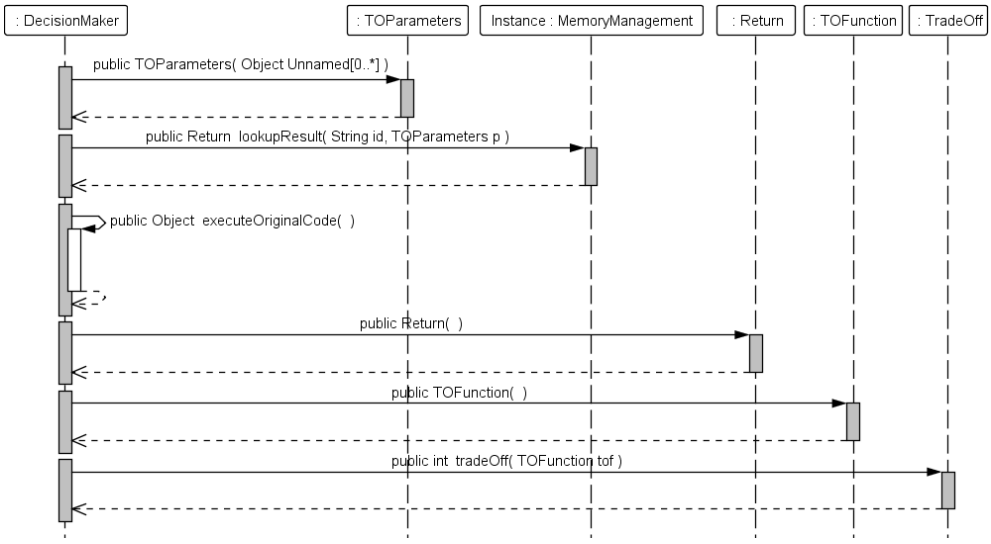


FIGURE 3.9 - LOOKUP EXECUTION FLOW - MISS

Another possible flow is that the data is not found (Miss) the Decision Maker receives a miss from the lookup method. Whenever it happens, this point is calculated and passed to the *TradeOff* module. In this case, an object *TOFunction* will be instantiated to keep all data and the free memory will be checked. If there is enough room, this point is added to the value table and the value 0 is returned to the Decision Maker. Supposing all the memory is full and the function is already tabulated. The score of that point is calculated

and compared to the last score of the tabulated values, if the new point score is higher than it is discarded and the value 1 is returned to the Decision Maker, otherwise the worst value, in terms of score, is deleted and the new one is added returning the value 0. The sequence diagram in figure 3.9 details this flow.

CHAPTER

4. BUSINESS INTELLIGENCE

In this chapter it is presented all the characteristics of the Business Intelligence module, including the explanation of the details of each analysis the application provides: energy, timing, memory and priority.

The Business Intelligence is actually a stand-alone dashboard application which has access only to the output data. The output data referred is generated by the other modules as they run concurrently with the system under analysis/profiling.

The main goal of the Business intelligence is to evaluate the data and provide a graphical presentation in a convenient way to help determine the efficiency of the architecture toward the target application. It will present some evaluations of memory usage and timing as well as the power consumption (considering power consumption models of CPU and memory components). Also, it needs the ability to forecast power consumption given changes in hardware or software configuration.

4.1 TIMING ANALYSIS

Timing analysis contributes to visualize how are the impacts of the function tabulation are over the relative and absolute execution time. Initially, the system estimates all the timings necessary to perform predictions. On the following table, it is presented all the timing data profiled from a test application:

Function Id	Frequency	One Exec. (ms)	Normal Exec. (ms)	Lookup (ms)	Best Exec. (ms)	Green Exec. (ms)	Speedup	α	α-be
Idle Time	-	-	4,501.77	-	4,501.77	4,501.77	-	-	-
Function 3	33942	0.601	20,393.75	0.279	12,898.06	12,898.06	36.76%	0.834	0.469
Function 1	27325	0.572	15,642.91	0.279	10,766.05	10,766.05	31.18%	0.801	0.492
Function 0	21785	0.547	11,920.25	0.279	8,471.78	8,471.78	28.93%	0.801	0.515
Function 2	25710	0.401	10,306.60	0.279	8,913.46	8,913.46	13.52%	0.834	0.7
Function 4	15117	0.591	8,932.08	0.279	7,076.23	7,076.23	20.78%	0.683	0.477
			71,697.34		52,627.34	52,627.34			

TABLE 4.1 - TIMING PROFILING RESULTS

The table's first column is the name of the function, followed by frequency that represents the number of calls the function received (N_{calls}). Then there is "One Exec." which is the average computation time for a single function call ($T_{computation}$). The fourth column is the estimate of the value of time spent by the functions not being tabulated, calculated by multiplying N_{calls}*$T_{computation}$. The next column represents the lookup time (T_{hit}) that is the time spent to retrieve a value during a hit. Then, "Best Exec." is the minimum amount of time possible considering the quantity of misses are equal to the number of parameters tabulated, while "Green Exec." is the actual time spent considering there was lack of memory to memorize values. "Best Exec." and "Green Exec." are respectively calculated by the following equations:

$$Ti_{best} = \left(T_{tradeoff} + T_{miss} + T_{computation}\right) * Ni_{parameters} + T_{hit} * Ni_{hit} \qquad (4.1)$$

$$Ti_{green} = \left(T_{tradeoff} + T_{miss} + T_{computation}\right) * Ni_{miss} + T_{hit} * Ni_{hit} \qquad (4.2)$$

The last three columns present come statistics of the functions. The speedup refers to the percentage of improvement between “Normal Exec.” and “Green Exec.”. Moreover, the BI calculates the current value of α and α_{be} explained in chapter 3.

To estimate the *Idle Time*, which corresponds to the amount of time the system were executing portions of code else than the tabulated functions; this value is important to determine the absolute speedup. It is calculated from the total time used during profiling decremented by the amount of time used by n-functions and the mechanism overhead accordingly to the equation:

$$T_{idle} = T_{total} - \sum_{i=1}^{n}\left[\left(T_{tradeoff} + T_{miss} + Ti_{computation}\right) * Ni_{miss} + T_{hit} * Ni_{hit}\right] \qquad (4.3)$$

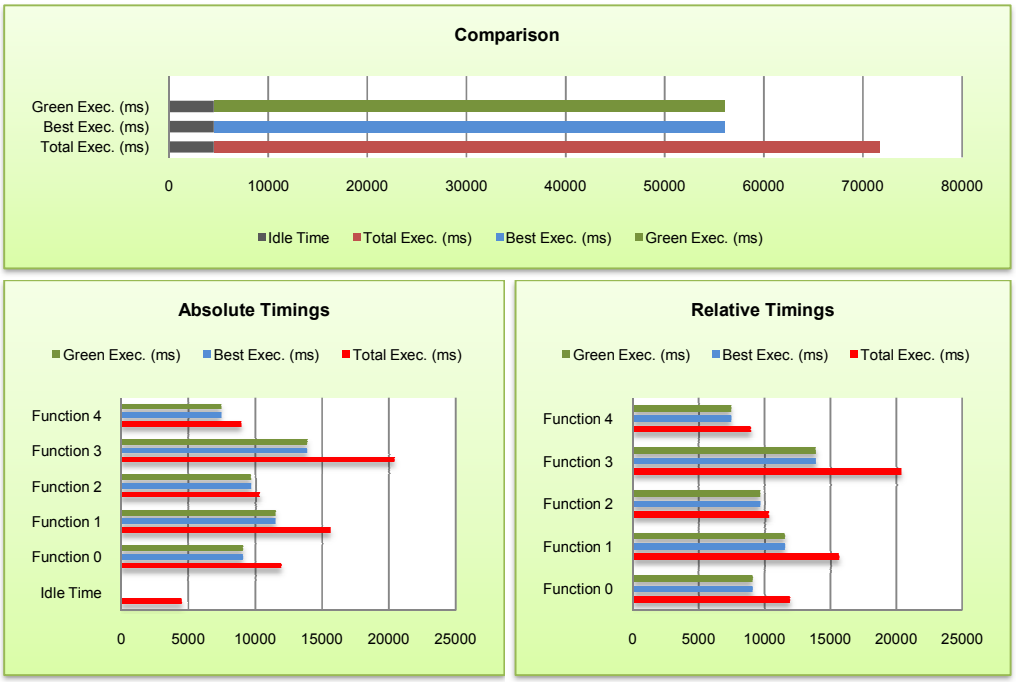


FIGURE 4.1 - TIMING SAMPLE CHARTS

To better understand how time is consumed, the dashboard charts the information. It is possible to visualize that “Best Exec.” and “Normal Exec.” work as lower and upper bounds to “Green Exec.” respectively.

4.2 MEMORY ANALYSIS

Memory analysis contributes to visualize how the memory is being used and managed within the Memory Management module. The main goal of this analysis is to enable the evaluation of possible scenarios, by varying the amount of memory reserved to tabulation and promote a forecast of the usage. i.e.:

	Size (Bytes)
Memory Size	268435456
Memory Usage – System	142981120
Memory Free – System	125454336
Memory Usage – Application	16252928
Memory Usage – Functions	5483296
Memory Free – Application	8355032

Function Id	# Parameters	Size (Bytes)	Memory (Bytes)
Function 0	4328	224	969472
Function 1	5433	224	1216992
Function 2	4277	224	958048
Function 3	5649	224	1265376
Function 4	4792	224	1073408
			5483296

TABLE 4.2 – MEMORY PROFILING RESULTS

The table on the left brings memory measures for each aspect, from the system to the application scope. From the application scope, Java runtime provides methods that measures memory usage and free memory. Although, Java is not able to measure memory information from processes outside the Java Virtual Machine (JVM), the architecture makes use of an auxiliary library [15].

- **Memory Size:** defines the total amount of hardware memory installed

- **Memory Usage – System**: defines the total amount of memory used by the system (i.e: Operating System, other processes);
- **Memory Free – System**: defines the total amount of free memory on the whole system;
- **Memory Usage – Application**: defines the total amount of memory being used by the application under evaluation;
- **Memory Usage – Functions**: defines the total amount of memory being used by the *MemoryManagement* to store functions parameters;
- **Memory Free – Application**: defines the total amount of free memory available in the JVM;

The table on the right presents the memory consumption calculated for each function. The value is computed from the number of parameters each function has times the amount of memory each parameter occupies. The size of the parameter ($S_{parameter}$) depends on the number of input arguments (N_{input}) a function possesses plus an overhead due to object representation in the Java environment. To calculate the size of each entry (bytes) into the *HashMap*, it is used the equation below:

$$S_{parameter} = 24 + 8 + 8 + 24 + N_{input} * (8 + 32) + 24 + 16 + 32 + 48 \quad (4.4)$$

For a single tabulation of a function with one input parameter, the *MemoryManagement* requires 224 bytes to store the value. The amount of memory needed for a stored value is derived from the space each element occupies in Java and was estimated according to the table:

Class TOParameters		Class Return		Object HashMap	
Class	24	Class	24	Entry	48
Members	8 + 8	Members	16		
Objects	24	One Value	32		
One Value	8 + 32				

TABLE 4.3 - MEMORY SIZE OF EACH ELEMENT USED IN TABULATION

To better understand how memory is consumed, the dashboard charts the information. It is possible to visualize the impact of tabulation against the entire memory usage and help to better parameterize the system to use memory more efficiently and without causing much disturbance to other applications and the system itself.

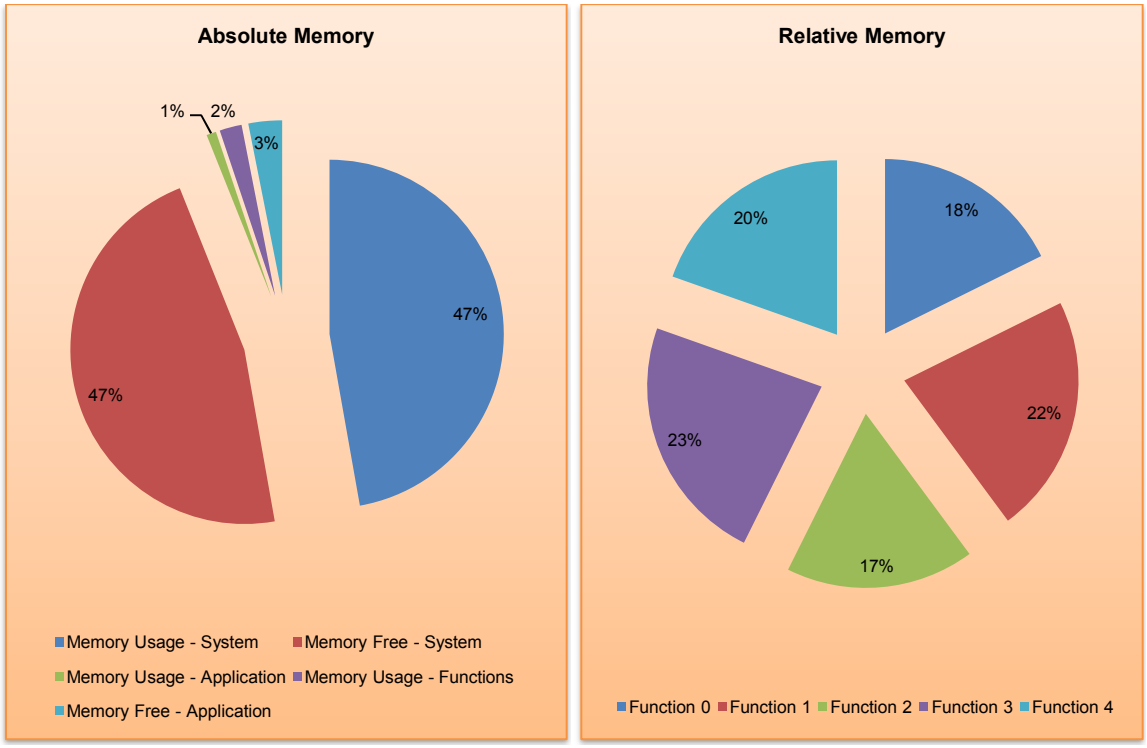


FIGURE 4.2 – MEMORY SAMPLE CHARTS

Memory analysis is important once memory itself is a limited resource and depends on the amount of physical memory modules installed and motherboard capacity. In addition, the correct estimation of memory use is important for what-if scenarios; increasing or decreasing the amount of memory results on impact to the capacity of memorization and consequently on timing due to the variation of miss rates.

4.3 ENERGY ANALYSIS

Energy analysis contributes to visualize how the energy is being spent regarding all the possible timing analysis and what-if scenarios of memory. The main goal of this analysis is to enable a comparison of each setup and provide savings report in terms of energy or money. i.e.:

Scenario	Memory (Wh)	CPU Idle (Wh)	CPU Normal (Wh)	CPU Green (Wh)	Savings (Wh)	CPU Best (Wh)	Max Savings (Wh)
Original Data	0.084	0.125	2.053	1.471	0.583	1.471	0.583
Memory - 16MB	0.084	0.125	2.053	1.471	0.583	1.471	0.583
Memory - 8MB	0.084	0.125	2.053	1.471	0.583	1.471	0.583
Memory - 4MB	0.084	0.125	2.053	1.683	0.371	1.471	0.583
Memory - 2MB	0.084	0.125	2.053	2.271	-0.218	1.471	0.583
Memory - 1MB	0.084	0.125	2.053	2.636	-0.583	1.471	0.583

TABLE 4.4 – ENERGY ESTIMATED RESULTS

The table above shows the results of energy consumption estimated to every configured scenario. The idea of scenarios, as seen before, comes to forecast changes in energy, memory and timing. To put more in details, each column is explained:

- **Memory**: this column represents the amount of energy spent to power the memory modules. It is estimated considering the total time spent during the profiling (T_{total}) and the power consumption of the module. Given that the power of a physical memory module is constant while in use or idle, the equation is:

$$E_{memory} = T_{total} * P_{memory} \quad (4.5)$$

- **CPU Idle**: this column represents the amount of energy spent to power the processor during the period it is not used to execute any of the tabulated functions or during any operation required by the *MemoryManagement* or *TradeOff*. It is estimated considering the idle time (T_{idle}) and the power consumption of the processor during inactivity, which is:

$$E_{cpu\ idle} = T_{idle} * P_{cpu\ idle} \quad (4.6)$$

- **CPU Normal**: this column represents the amount of energy spent to power the processor during the period it is used to execute the tabulated functions but considering they are not tabulated and executing the normal flow. It is estimated summing the time of each function ($T_{computation}$) multipliedby the power consumption of the processor during activity, which is:

$$E_{cpu\ normal} = \sum_{i=0}^{N} Ti_{computation} * P_{cpu\ active} \quad (4.7)$$

- **CPU Green**: this column represents the amount of energy spent to power the processor during the period it is used to execute the tabulated functions considering the actual flow of the memorization architecture. It is estimated summing the time of each function in green operation (T_{green}) multiplied by the power consumption of the processor during activity, which is:

$$E_{cpu\ green} = \sum_{i=0}^{N} Ti_{green} * P_{cpu\ active} \quad (4.8)$$

- **Savings**: this column represents the amount of energy saved by using the tabulation mechanism. It considers the difference between the “CPU Normal” and “CPU Green”. It correspond to the actual energy saved by means of tabulation:

$$E_{savings} = E_{normal} - E_{green} \quad (4.9)$$

- **CPU Best**: this column represents the amount of energy spent to power the processor during the period it is used to execute the tabulated functions considering the best possible flow of the memorization architecture. It is estimated summing the time of each function in best operation (T_{best}) multiplied by the power consumption of the processor during activity, which is:

$$E_{cpu\ best} = \sum_{i=0}^{N} Ti_{best} * P_{cpu\ active} \quad (4.10)$$

- **Max Savings**: this column represents the amount of energy saved by using the tabulation mechanism. It considers the difference between the "CPU Normal" and "CPU Best". It correspond to the best possible scenario for energy savings by means of tabulation:

$$E_{savings} = E_{normal} - E_{best} \quad (4.11)$$

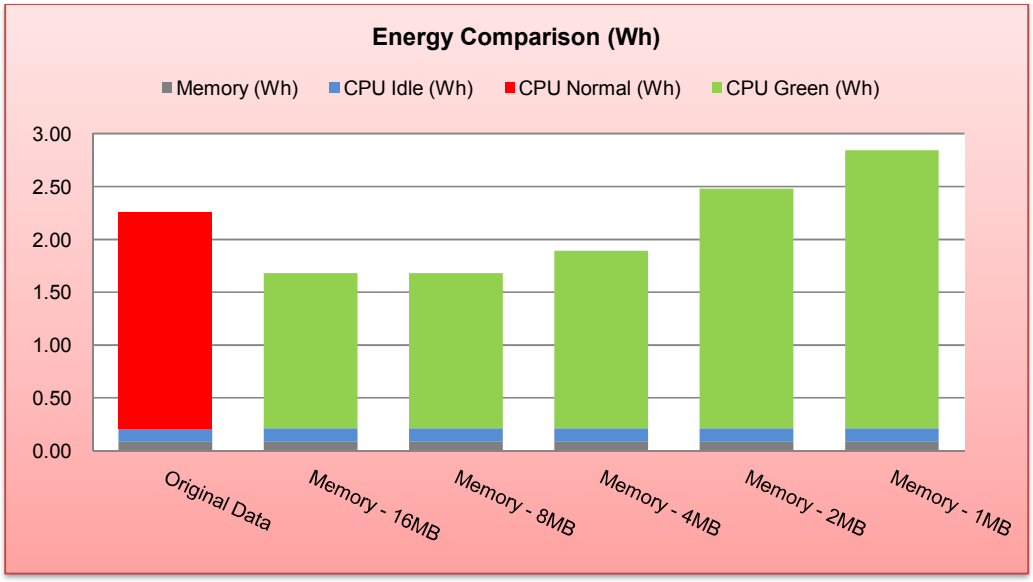


FIGURE 4.3 – MEMORY ESTIMATED CHARTS

The previous chart shows the comparison between every configured scenario against the original profiling data. As one can immediately spot that memory directly influences the energy consumed and increase when it is restricted. The reason for it is the increase of the miss rate, once stored parameters have to be discarded and increases the number of time the function has to be calculated plus the overhead code of the *DecisionMaker*.

4.4 FUNCTION ANALYSIS

Function analysis contributes to visualize how the *MemoryManagement* prioritizes functions during tabulation. It is important to notice that the priority weights the amount of memory distributed to each function, which means, depending on the function's priority the higher the priority the higher the amount of memory it has reserved for memorization. Originally, the architecture considers only timing information to calculate priority; the business intelligence proposes new forms of memory distribution according to timing, memory usage, power consumption. i.e.:

Function Id	Normal Timing (%)	Green Timing (%)	Best Timing (%)	Memory (%)	Parameter Size (%)	Scenario Priority	System Priority
Function 3	30.350	26.801	26.801	23.077	20.000	0.267	0.304
Function 1	23.280	22.371	22.371	22.195	20.000	0.213	0.233
Function 0	17.740	17.603	17.603	17.680	20.000	0.204	0.177
Function 2	15.338	18.521	18.521	17.472	20.000	0.178	0.153
Function 4	13.293	14.704	14.704	19.576	20.000	0.138	0.133

TABLE 4.5 – FUNCTION STATISTICS RESULTS

The table above illustrates how each aspect of the system behavior is related to each other. The data is weighted to represent the proportion of each functions has to the total value. The idea is to let the user free to configure different priorities considering different combinations of each weight. For instance, the "System Priority" (PR_{system}), which is the current implementation of priorities, is calculated using "Normal Timing" only; the "Normal Timing" corresponds to the time consumed by each function before the tabulation mechanism is applied:

$$PRi_{system} = \frac{Ti_{normal}}{\sum_{j=0}^{N} Tj_{normal}} \tag{4.12}$$

Although, this approach may not be the best considering there are other parameters such as memory that could influence a priority. As so, one example could be by combining

the “Normal Timing” and the inverse value of “Memory”, which therefore prioritizes functions that requires more time while using less memory:

$$PRi_{scenario} = \frac{\frac{Ti_{normal}}{\sum_{j=0}^{N} Tj_{normal}} * \frac{\sum_{j=0}^{N} Mj_{normal}}{Mi_{normal}}}{\sum_{j=0}^{N} PRj_{scenario}} \quad (4.13)$$

As the priority are normalized, it can be used to distribute the amount of memory every function deserves by weighting the total available memory with the priority value.

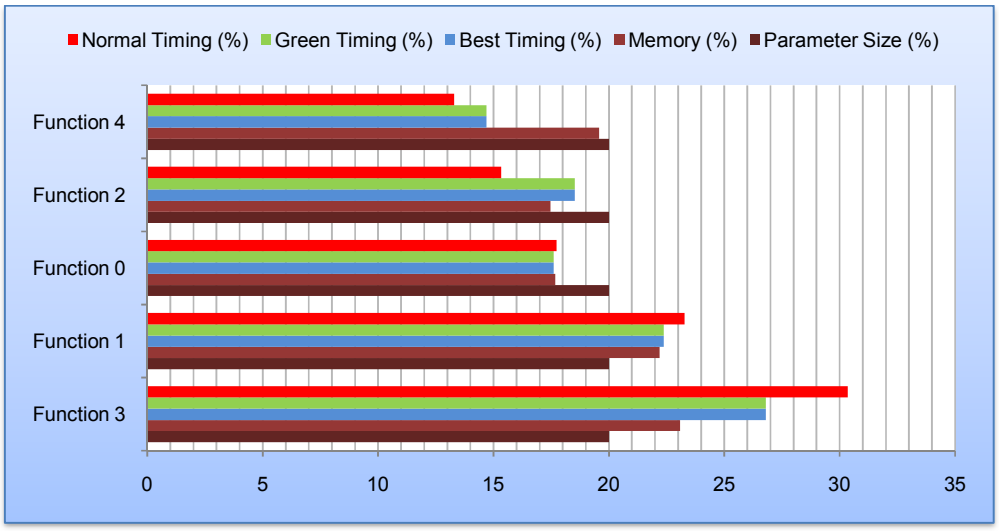


FIGURE 4.4 – FUNCTION STATISTICS CHARTS

The previous chart shows the comparison between every weight component that can be used to compose the priority. Although, one cannot tell immediately what is the best set of weights to compose a good priority and for that reason we let it to be determined by simulation.

4.5 SCENARIOS

The business intelligence module is developed to support the creation of what-if scenarios. What-If scenarios are used to examine how the outcome of a given problem would be having different input parameters or configuration. Applying this methodology to the Green IT perspective, it would be interesting to evaluate how different configurations of hardware (i.e.: processor, memory) and software (i.e.: priority, function scope) perform in terms of final energy efficiency.

The next sections are going to explain how what-if scenarios can be configured in the dashboard application and what the expected impacts of each approach are:

4.5.1 PRIORITY

Priority, as seen before, defines the preference a function have over the others. The visible impact of priority comes when there is limited memory for tabulation. In this case, every new call to a function, considering the parameter set is not yet tabulated, may require that a prior stored value to be removed. Considering only high priority functions, it is expected of them to take over the memory space, eliminating all the other functions.

This effect is probably intensified when a simple priority is defined. The underlying idea is that only one aspect (timing, memory) would not define a best possible solution. As an example, we can say that not only how much time a function is expected to use from the total amount but also if this function uses less memory to do so – “Normal Timing” times the “Parameters size” inversed could be a good fit.

From the function analysis in section 4.4, the dashboard application provides a set of possible ways to configure the priority, by using the following definitions:

- **Tnormal / iTnormal**: Is the percentage of each function regarding Normal Execution Time and its inverse value;
- **Tgreen / iTgreen:** Is the percentage of each function regarding Green Execution Time and its inverse value;
- **Tbest / iTbest**: Is the percentage of each function regarding Best Execution Time and its inverse value;
- **Mfunction / iMfunction**: Is the percentage of each function regarding memory occupied by a functions parameters and its inverse value;
- **Mparameter / iMparameter**: Is the percentage of each function regarding memory occupied by a single parameter and its inverse value;

4.5.2 MEMORY

Memory space has direct impact on the system performance. As more parameters set are tabulated, the bigger is the minimization of the overhead time spent by the tabulation mechanism. The goal of memory being a parameter for what-if scenarios is to predict how the system would execute considering increase and decrease of memory space for memorization.

Additionally, it is also important that *MemoryManagement* provides enough capabilities to enable the distribution of memory regarding priority. Considering policies of memorization during reducing memory space, there are three possibilities:

- **Naïve**: based on the priority, every time there is need for space the memory removes a random parameter from the lowest priority function. This lead to a situation where the lowest priority function could be completely removed from the memory.

- **Ordered**: it is basically the same as naïve, but in this case parameters are removed from the lowest to the highest frequency. As the frequency is incremented to the miss quantity (N_{miss}), the effects of the overhead is tried to be postponed at most.
- **Even**: having in mind that even if low priority functions have less achieved efficiency, it might happen that parameters from low priority functions but with high frequency are more effective than high priority functions but with low frequency parameters. Therefore, the idea is to discard parameters proportionally among all functions.

Considering policies of memorization during increasing memory space, it is necessary to have more information about the limits of each function input parameter and its precision. By doing so, it is possible to estimate the amount of parameters a function would have at most and then by comparing to the profiled amount determine the percentage of the function is memorized. By then, we can simulate by increasing the number of tabulated parameters in each function proportionally to the priority and assign frequencies that follows the mean and variance of the current frequency table.

Ultimately, it is common sense that by changing the hardware configuration implies in different power requirements, which is, increasing or decreasing the number and type of physical memory banks we have different energy consumption. As so, the user has to provide information about the characteristics of the memory modules used and its power consumption. Vide appendix A.

4.5.3 CENTRAL PROCESSING UNIT

The processor or CPU is not referred before on the methodologies, but it has a fundamental impact on the system performance. At the end of the day, the amount of time needed to execute every piece of code is a function of the CPU capabilities. Furthermore, it

can present different energy consumptions according the technology used to build as well as the architecture (CISC, RISC).

As so, the dashboard application provides a simple support to configuration regarding CPU power and time. Considering the processor has different power requirements during idle and active periods, one could configure scenarios with for different types of processors. Also, it is available a correction factor for the timing purposes, so for a different CPU regarding the one used to profile, time could be corrected before evaluation. Vide appendix A.

4.6 IMPLEMENTATION

The Business Intelligence dashboard application was built using Java programming language and NetBeans as development environment. Moreover, all the evaluations performed are done at the same classes the actual tabulation mechanism uses, that is, the dashboard application is fully integrated with the common classes.

The application is designed to be a stand-alone application, and could run in different computers considering that the profiling data is available. Once the dashboard does not interfere in the running architecture and no real time requirements, performance is not an issue. The following paragraphs are going to describe the classes of the report package.

Classes in the report package are related to the support of the Graphical User Interface (GUI). The entry point is the *BusinessIntelligence* class that controls the instance of *BusinessIntelligenceView*. There are also separate classes that render each different analysis (*DashBoardTimingPanel*, *DashBoardMemoryPanel*, *DashBoardEnergyPanel*,

DashBoardFunctionPanel) and all implement the interface *IDashBoardChartPanel* used to update the charting renderization.

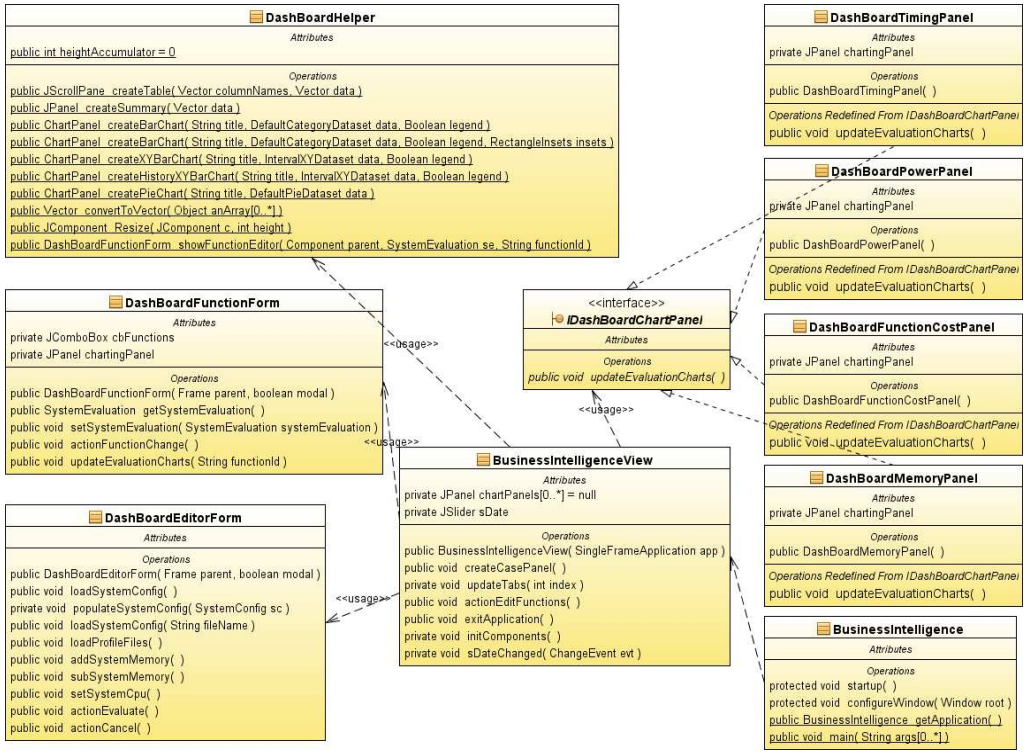


FIGURE 4.5 – CLASS DIAGRAM OF REPORT PACKAGE

The class *DashBoardHelper* provides functions to facilitate the creation of charts and to integrate the dashboard to the charting library [16]. Finally, *DashBoardFunctionForm* and *DashBoardEditorForm* enable some further configuration of the application parameters and scenarios.

Classes in the report.entities package are related to the support of the evaluation of the profiling data and the simulation of every scenario. The main class is *SystemEvaluation* which aggregates the data and provide methods to every information available on the interface (charts, tables). To each scenario, there is a *SystemConfig* that interfaces the configured memory and CPU, *SystemMemory* and *SystemCpu* respectively.

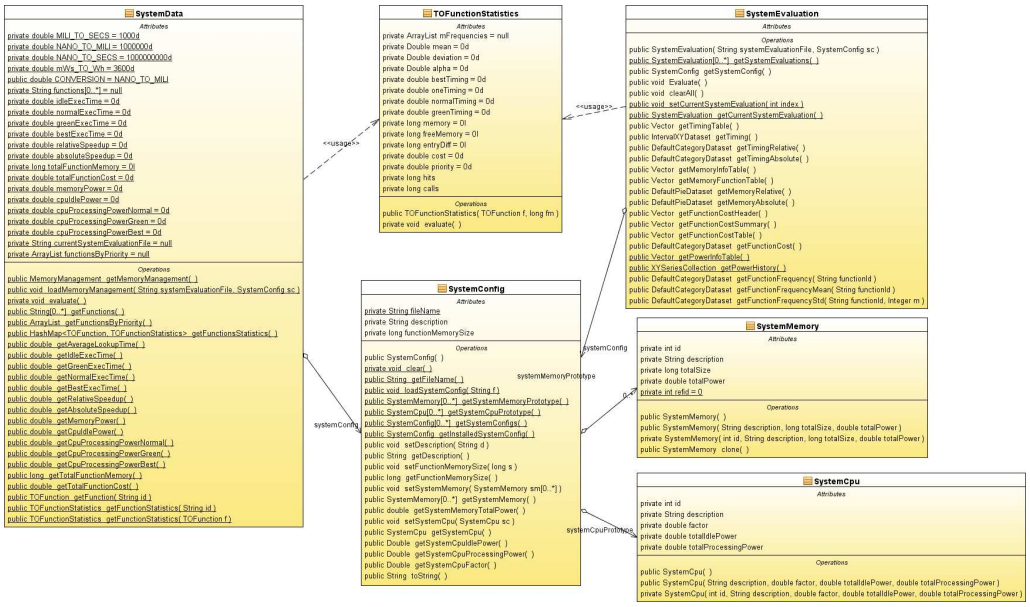


FIGURE 4.6 - CLASS DIAGRAM OF REPORT.ENTITIES PACKAGE

Given that, the class SystemData collects the data from the MemoryManagement restored from profile file and calculates all the statistics regarding a single *SystemConfig* . Also, each function is adapted to a new class *TOFunctionsStatistics,* from the original *TOFunction* , to make it easy to work and to provide further statistical information.

CHAPTER

5. BUSINESS INTELLIGENCE RESULTS

In order to validate the information provided on the dashboard, this chapter is going to present the evaluation of results from the profiling of real mathematical functions. Our main focus will be the tabulation applied to methods used on most of financial institutions.

Then we are going to present different scenarios of memory and priority in order to determine the best combination of parameters.

5.1 PRIORITY EVALUATION

To determine how priority interferes in the system performance and final results, we are going to consider a sample system that will serve as a demonstration only. Let's define a system with the following hardware configuration given on table 5.1. Moreover, we measure some of the system timing information for each tabulated function; in this case, we have defined 5 functions with following characteristics:

Function Id	Frequency	One Exec. (ms)	Normal Exec. (ms)	Green Exec. (ms)	α	α-be	Parameter Size (bytes)
Function 1	183833	1.546	284,214.64	108,364.49	0.923	0.212	344
Function 2	178997	1.556	278,470.82	105,647.15	0.923	0.21	344
Function 0	162500	1.538	249,911.68	95,687.80	0.923	0.213	384
Function 3	194166	1.019	197,806.03	105,707.13	0.929	0.348	224
Function 4	173685	0.976	169,438.92	93,380.75	0.933	0.368	344
			1,244,365.23	573,310.45			

System Execution Time (ms)	573,310.45	Memory Power	5.766 W
Average Hit Time (ms)	0.486	Cpu Idle Power	100 W
Average Miss Time (ms)	0.270	Cpu Active Power	110 W
Average Trade Off Time (ms)	0.014		

TABLE 5.1 – TABLE OF SAMPLE SYSTEM UNDER EVALUATION

The next step is to try different combinations of priorities and evaluate which gives the best results. As a method to compare how each priority performs, we are going to use the amount of energy savings (Wh) in each scenario. The following table presents a set of best performers considering the system we defined:

Memory Size (MB)	Ordered Policy (Wh)	Even Policy (Wh)			
		Tnormal	Tnormal, iMparameter	Mfunction	Tnormal^4
128	-	138.009	137.205	135.321	148.044
64	-	68.327	67.905	66.953	73.291
32	-	33.466	33.263	32.79	35.583
(Original Data) 20.50	20.504	20.504	20.504	20.504	20.504
16	15.995	16.846	16.671	16.665	16.463
8	4.925	6.683	6.65	6.497	6.711
4	-1.324	0.245	0.258	0.135	0.362
2	-3.931	-3.426	-3.417	-3.493	-3.34
1	-5.631	-5.454	-5.443	-5.486	-5.392

TABLE 5.2 – TABLE OF ENERGY SAVING FROM DIFFERENT PRIORITIES

When memory size is from 1 MB to 16 MB, lower with respect to the amount of memory occupied from the original data (20.5MB), the dashboard evaluation class starts removing parameters from the memorization table according to the priority. Although when

the policy is *Ordered*, it does not matter how the priority is composed once parameters are removed from the lowest priority to the highest priority function, where the order is always the same.

With respect to policies that evenly distribute memory, we can verify how they impact energy savings when limiting memory, but also, to predict how much energy is going to be saved when the memory is increased. To forecast how function parameters are distributed, we consider range and precision which enables to estimate how many parameters a function could possess. In this case, we increase the number of parameters and assign frequencies according to the distribution model.

From the table we verify that *Tnormal* has a good performance when reducing memory while *Tnormal^4* provide better results when memory is expanded. The interesting fact is that the parameter size and how much memory a function uses has low or null improvement toward a better priority. As we can see in <*Tnormal,iMparameter*>, the results of every memory scenario has less energy savings than when using only *Tnormal*. Having that in mind, not only *Tnormal* constitutes the best priority as so as it is the simpler to implement, which would reduce any overhead due to complexity of implementation and data acquisition.

5.2 FINANCIAL SOFTWARE

Nowadays, several financial institutions use complex systems to manage all its resources, including fees, taxes, accounts, investments and prices in general.

These systems are well-known as heavy computational operations including libraries and routines used to calculate various different values related to personal and corporate economies and accounting at all. They are complex because use math with

exponential, rational numbers and series. Obviously, as the most of managerial systems, a set of variables and changeable parameters are available to the user, and this turn the servers into operations which if not run in high performance equipments may take several amounts of time.

This work will focus in software layer of Green IT field and furthermore, will take place in systems related to financial operations and calculations. Analyzing operations and using software profiles to verify and identify routines, it will be possible to optimize such executions, release idle resources and probably improving the all available resources.

Initially the work will take place in analysis of a set of functions traditionally used in system to manage credit and insurance companies. They are:

- **Implied Volatility**: of an option contract is the volatility implied by the market price of the option based on an option pricing model. In other words, it is the volatility that, when used in a particular pricing model, yields a theoretical value for the option equal to the current market price of that option. Non-option financial instruments that have embedded optionality, such as an interest rate cap, can also have an implied volatility. Implied volatility, a forward-looking measure, differs from historical volatility because the latter is calculated from known past returns of a security;
- **Binomial Option Pricing Model**: is widely used as it is able to handle a variety of conditions for which other models cannot easily be applied. This is largely because the BOPM is based on the description of an underlying instrument over a period of time rather than a single point. As a consequence, it is used to value American options that are exercisable at any time in a given interval as well as Bermudan options that are exercisable at specific instances of time. Being relatively simple, the model is readily implementable in computer software;

These calculations presented above are also considered the algorithmically bottleneck of Financial Enterprise Systems, compromising overall performance. In fact, to attack those points, it will be necessary to identify the pure functions, in other words, mathematically pure functions (functions without any other functionality but math no output, no interface, no event handling).

The study regards some substitutions of Series calculation by Tables with pre calculated values depending of chosen parameters to optimize time of financial software execution.

5.2.1 ENERGY SAVINGS

To determine the system prediction performance, we executed the tabulation architecture against the financial functions defined: Implied Volatility, Binomial Option Pricing Model. The idea to limit the memory size and then execute the system with a fixed amount of calls for each function. The system will then run toward a steady state. The we compare the predicted value of savings the dashboard will provide considering memory dump of each scenario. The following table presents the data collected regarding energy savings

Scenario	Actual	From 2048KB	From 1024KB	From 512KB	From 256KB	From 128KB	From 64KB	From 32KB
2048	**7.097**	**7.097**	5.075	5.015	4.858	4.850	4.725	4.961
1024	**5.024**	4.755	**5.024**	4.694	4.316	4.209	3.99	4.243
512	**4.165**	4.083	4.445	**4.165**	3.532	3.290	3.043	3.278
256	**2.568**	2.821	3.208	3.228	**2.568**	2.283	2.045	2.213
128	**1.379**	1.570	1.945	1.906	1.839	**1.379**	1.174	1.329
64	**0.588**	0.654	1.052	0.926	0.968	0.901	**0.588**	0.688
32	**0.301**	0.102	0.505	0.342	0.396	0.422	0.369	**0.301**

TABLE 5.3 – ENERGY SAVING RESULTS FROM ACTUAL AND PREDICTION DATA

In the end, energy savings depends only on the difference of processor power between idle and active states. In this case, the values may change for different processors but the proportions will be kept.

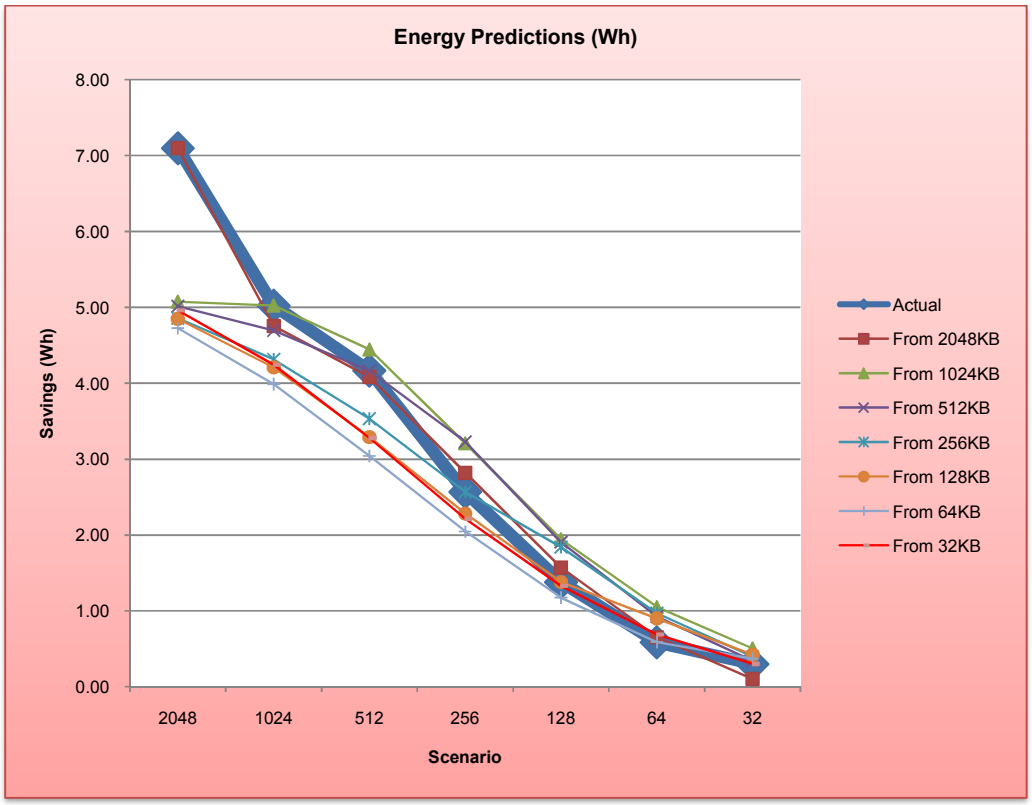


FIGURE 5.1 - ENERGY SAVING RESULTS FROM ACTUAL AND PREDICTION DATA

In the figure 5.1, the bold line represents the actual savings achieved from running the system. The thin lines are predictions according to the data restored from each execution.

As expected, prediction data is more accurate as we have more information of the system behavior, that is, when there is more memory space, more function data is tabulated as well as their frequency information. Considering prediction data for 2048KB, we can verify a big gap which is due to unbalanced memory distribution, that is, memory was not

divided accordingly to the functions priorities. In addition, timing results ($T_{computation}$, T_{miss}, T_{hit} and $T_{tradeoff}$) will slightly change in every execution. Even though, the final results are satisfactory.

CHAPTER

6. CONCLUSION

Green IT is a research field becoming even more important nowadays when natural resources are considered important to sustainable development. With importance increasing and researches, new technologies are starting to appear in the direction of natural resources saving and environmental impact avoidance. Since energy is directly related to Green impact, and beside is also becoming an expensive attribute in many companies' budget, the Green IT is trying to find the least intrusive answer to already existent solutions. Why?! Because modifying equipments and change systems running currently in industry organizations are very expensive and may not guarantee the reliability and safety previously analyzed.

Then, taking into account companies with huge datacenters that provide computational power over business information, this work started focusing in software layer which is an very customizable field and also flexible to changes. Previous researches had demonstrated that the way the software layer is executed in huge datacenters with high data load impact differently in the overall hardware behavior by means of microprocessor power consumption, amount of memory banks powered permanently and also the generated heat. Heat, a very important issue in Green IT and datacenters configuration, is responsible for a

large slice of the power consumption in IT since the air conditioning systems are usually high energy consumers.

This work, the Business Intelligence Module, is part of a larger project that aims to develop architecture to support function tabulation and which includes Pure Function Recognition, Bytecode Modification, Trade-Off and Memory Management. The Business Intelligence Module is an external tool to better evaluate and understand how the rest of the architecture is performing. The implementation has been done in Java.

It is verified that tabulation is best when there is infinite memory space and priority has no effect. Although there is I no such thing as infinite memory, the first set of results is regarding the priority mechanism. For simplicity, the current implemented priority mechanism only takes into account the original computational timing to determine it. The results confirmed that the use of the function total computational time is the best performer in terms of energy saved, plus the data for calculating this priority is easier comparing to other data.

Now regarding the acuity of the predictions made by the application, as expected, prediction data is more accurate as we have more information of the system behavior, that is, when there is more memory space, more function data is tabulated as well as their frequency information. Even though, timing results ($T_{computation}$, T_{miss}, T_{hit} and $T_{tradeoff}$) will slightly change in every execution the final results are satisfactory.

6.1 FUTURE WORK

Having in mind the promising results obtained in this project, there is a set of possible enhancements to be put in practice in subsequent work steps. The future work to be develop is to integrated static function analysis from the pure function module, even

more, some totally new information like: cpu usage, parameter range, parameter precision and so on.

Other possible improvement is regarding the power model. It should be validate through physical tests and made as accurate as possible by determining what kind of characteristic should be taken into account when calculating the final value. Moreover, as known that cpu generates heat according to its usage level, the power model could consider how energy consumption is reduced from the point of view of cooling requirements.

REFERENCES

[1] ISO/IEC, TR9126:2003, Software engineering - Product quality, International Organization for Standardization, Geneva, Switzerland., 2003.

[2] C. Francalanci, E. Capra, and G. Agosta. Developing Energy-Efficient Software: Enersoft., 2009.

[3] T. Restorick. An inefficient truth. global action plan. IT Professional, 2007.

[4] E. Williams. Energy intensity of computer manufacturing: Hybrid assessment combining process and economic input-output methods. Environ. Sci. Technol., (38):6166–6174, 2004.

[5] Berkeley National Lab. Lawrence. Optimization of product life cycles to reduce greenhouse gases in california. Report for California Energy Commission., (CEC-500-2005-110-F), 2005.

[6] Industrial Research and Development Corporation. Personal computers (desktops and laptops) and computer monitors. Report for the European Commission, August 2007., 2007.

[7] Kenneth Cayton and Jed Scaramella. Ibm system x4 : Delivering high value through scale up. Technical report, IDC, 2008.

[8] Brown E.G. and JLee C. Topic overview: Gren it. Technical report, Forrester Research, 2007.

[9] http://www.tpc.org.

[10] http://www.ambiente.it/impresa/legislazione/leggi/2005/digs151-05.htm.

[11] N. Margolus and L.B. Levitin. The maximum speed of dynamical evolution. Phisica, D120:pp. 188–195, 1998.

[12] C. Francalanci and E. Capra. Green it. sfide e opportunit`a. http://www.mondodigitale.net/Rivista/08_numero_4/Francalancip.36-42_.pdf, 2007.

[13] San Murugesan. Harnessing green it: Principles and practices. *IT Professional*, vol.10 no.1:pp. 24-33, 2008.

[14] Matthew Finifter, Adrian Mettler, Naveen Sastry, and David Wagner.Veri_able functional purity in java. In CCS '08: Proceedings of the 15th ACM conference on Computer and communications security, pages 161-174, 2008.

[15] http://github.com/jezhumble/javasysmon/

[16] http://www.jfree.org/jfreechart/

APPENDIX A

The following XML file corresponds to a possible dashboard configuration file (SystemConfig.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Document   : SystemConfig.xml.xml
    Created on : March 11, 2010, 11:59 PM
    Author     : planer
    Description: Configures the system properties
        Purpose of the document follows.
-->
<config>
  <!--
        <memory id="{int}" description="{String}" totalsize="{long:Bytes}" totalpower="{double:Watts}" />
        totalsize: describes the size of memory used.
        totalpower: describes the power spent. Used to calculate energy over time.
    -->
  <memory id="1" description="Memory 128MB" totalsize="134217728" totalpower="0.24025" />
  <memory id="2" description="Memory 256MB" totalsize="268435456" totalpower="0.4805" />
  <memory id="3" description="Memory 512MB" totalsize="536870912" totalpower="0.961" />
  <memory id="4" description="Memory   1GB" totalsize="1073741824" totalpower="1.922" default="1" />
  <memory id="5" description="Memory   2GB" totalsize="2147483648" totalpower="3.844" default="1" />
  <memory id="6" description="Memory   4GB" totalsize="4294967296" totalpower="7.688" />
  <memory id="7" description="Memory   8GB" totalsize="8589934592" totalpower="15.376" />
  <!--
        <cpu id="{int}" description="{String}" totalpower="{double:Watts}" />
        factor: describe the efficiency of the CPU comparing to the default/measured CPU.
        totalidlepower: describes the power spent during idle operation.
                Used to convert from processing time(s) to power.
        totalprocessingpower: describes the power spent during processing operation.
                Used to convert from processing time(s) to power.
    -->
  <cpu id="1" description="AMD Athlon XP 2700+" factor="0.9" totalidlepower="90" totalprocessingpower="9" />
  <cpu id="2" description="AMD Athlon XP 3200+" factor="1.1" totalidlepower="110" totalprocessingpower="11" />
  <cpu id="3" description="AMD Athlon XP 2700+" factor="1.2"  totalidlepower="120" totalprocessingpower="12" />
  <cpu id="4" description="Intel Core2Duo 2.1GHz" factor="1.0" totalidlepower="100" totalprocessingpower="10"
default="1" />
  <!--
        <evaluation description="{String}" functionmemorysize="{long:Bytes}" />
        functionmemorysize: amount of memory dedicated to function memorization.
        installed:especifies the scenario configuration where profiling was done.
    -->
  <evaluation description="Original Data" installed="1">
    <priority compose="Tgreen,iMfunction"/>
    <cpuref refid="4" />
    <memoryref refid="4" />
    <memoryref refid="5" />
  </evaluation>
  <evaluation description="Memory - 4MB - Ordered" functionmemorysize="4194304" >
    <priority compose="Tgreen,iMfunction" policy="ordered" />
    <cpuref refid="4" />
    <memoryref refid="4" />
    <memoryref refid="5" />
  </evaluation>
</config>
```

APPENDIX B

The following table presents the current DTD validation to the *MemoryManagement* dump and a sample piece of file from a real profiling execution.

```
<!-- DTD definition -->
<?xml version="1.0" encoding="UTF-8"?>
<!-- Root Node -->
<!ELEMENT MemoryManagement (information, functions)>
<!ELEMENT information (info+)>
<!ELEMENT info (#PCDATA)>

<!ELEMENT functions (function+)>
<!ELEMENT function (timing, frequency, hits,  memory, priority, frequencies)>
<!ELEMENT timing (#PCDATA)>
<!ELEMENT frequency (#PCDATA)>
<!ELEMENT hits (#PCDATA)>
<!ELEMENT memory (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
<!ELEMENT frequencies (frequency+)>

<!-- Function Attributes -->
<!ATTLIST function signature CDATA #REQUIRED>
<!-- Info Attributes -->
<!ATTLIST info name CDATA #REQUIRED>
<!ATTLIST info type CDATA #REQUIRED>
<!ATTLIST info value CDATA #REQUIRED>
<!-- Data Attributes -->
<!ATTLIST frequency name CDATA>
<!ATTLIST frequency type CDATA>
<!ATTLIST frequency value CDATA #REQUIRED>
<!ATTLIST timing type CDATA #FIXED "Long">
<!ATTLIST timing value CDATA #REQUIRED>
<!ATTLIST memory type CDATA #FIXED "Long">
<!ATTLIST memory value CDATA #REQUIRED>
<!ATTLIST priority type CDATA #FIXED "Float">
<!ATTLIST priority value CDATA #REQUIRED>
<!ATTLIST entries type CDATA #FIXED "Long">
<!ATTLIST entries value CDATA #REQUIRED>
```

```
<!—XML Exemple -->
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<MemoryManagement date="2010-06-23 05:41:54">
  <information>
    <info name="systemExecTime" type="Long" value="146513485911"/>
    <info name="lookupAverageHitTime" type="Double" value="279127.460513073"/>
    <info name="lookupAverageMissTime" type="Double" value="257300.179745903"/>
    <info name="tradeoffAverageTime" type="Double" value="27096.0166673471"/>
    <info name="memorySize" type="Long" value="3215851520"/>
```

```
    <info name="memoryUsageSystem" type="Long" value="2337640448"/>
    <info name="memoryUsageApplication" type="Long" value="109117440"/>
    <info name="memoryUsageFunctions" type="Long" value="1312288"/>
    <info name="memorySystemFree" type="Long" value="878211072"/>
    <info name="memoryApplicationFree" type="Long" value="83100184"/>
  </information>
  <functions>
    <function methodName="binomialOptionPricing" signature="binomialOptionPricing">
      <timing type="Long" value="2249919"/>
      <frequency type="Long" value="498670"/>
      <hits type="Long" value="498670"/>
      <memory type="Integer" value="424"/>
      <priority type="Float" value="0.1864267"/>
      <frequencies>
        <frequency name="100.0+120.0+1.0+1000+0.032+0.28" value="13"/>
        <frequency name="100.0+120.0+1.0+1000+0.081+0.61" value="1"/>
        <frequency name="100.0+120.0+1.0+1000+0.059+0.76" value="17"/>
        <frequency name="100.0+120.0+1.0+1000+0.078+0.77" value="1"/>
        <frequency name="100.0+120.0+1.0+1000+0.072+0.46" value="75"/>
        .
        .
        .
      </frequencies>
    </function>
  </functions>
</MemoryManagement>
```