Analyzing Programming Languages' Energy Consumption: An Empirical Study[*]

Extended Abstract[†]

Stefanos Georgiou[‡]
Athens University of Economics and Business
Athens, Greece
sgeorgiou@aueb.gr

Diomidis Spinellis[§]
Athens University of Economics and Business
Athens, Greece
dds@aueb.gr

Maria Kechagia[¶]
Delft University of Technology
Delft, The Netherlands
m.kechagia@tudelft.nl

ABSTRACT

Motivation:
Goal:
Method:
Results:

CCS CONCEPTS

• **Hardware** → **Power estimation and optimization**; • **Software and its engineering** → *Software libraries and repositories*; *Software design tradeoffs*;

KEYWORDS

GreenIT, Energy Efficiency, Energy Optimization, Programming Languages

ACM Reference format:
Stefanos Georgiou, Diomidis Spinellis, and Maria Kechagia. 2017. Analyzing Programming Languages' Energy Consumption: An Empirical Study. In *Proceedings of Panhellenic Conference on Informatics, Larrisa, Greece, September 2017 (PCI '17),* 3 pages.
https://doi.org/10.475/123_4

1 INTRODUCTION

The increase demands of services and computational applications from ICT-related products are the major facts contributing to increase energy consumption.[1] Recent results obtained by Gelenbe and Caseau [4] and Van Heddeghem et al. [7] indicates the raising course of the IT's sector energy requirements, which are expected to reach 15% of the world's total energy consumption by 2020. Moreover, recent results outline, also, raise of green house gas emissions due to the IT-sector which are growing much faster than initially predicted and are estimated around 2.3% globally, as claimed in SMARTer2030[2] report. Therefore, providing energy efficiency at different fields of IT is essential and of paramount importance, since the fact does not only contributes economically but also environmentally.

Traditionally, most of the studies, regarding energy efficiency, considered energy consumption at hardware level. However, nowadays, there is much of evidence that software can also alter energy dissipation significantly [1–3]. In fact, some researchers showed by using inefficient design patterns or data structures can increase energy dissipation up to 300–700% [5, 6]. Nevertheless, even by applying minor modifications in application's energy usage can contribute to large scale changes if taken into account the amount of available PC-like mobile devices and data-centers' servers.

In order to reduce energy consumption in software development, we conducted an empirical study aiming to elicit energy usage results from small tasks implemented in a variety of programming languages. Prior work only focused an a small number of programming languages which are mostly used in Android development such as Java and JavaScript or C and C++, by making use of Native Development Kit[3] library. Therefore, our aim in this research is to identify which programming languages offer more energy efficient implementations for different tasks.

The remainder of this paper is organized as follows. In Section 2, we discuss prior work done in the field and compare it with ours. Section 3 describes in details our experimental platform, the software and the hardware tools we used, how we refined our dataset, and our methodology for retrieve our results. In Section 4, we provide a discussion based on our preliminary results and Section 5 details threats of validity. Finally, we conclude in Section 6 and we discuss future work and possible directions for this research.

2 RELATED WORK

3 EXPERIMENT SETUP

In this Section, we describe the experimental approach we used in order to perform our research and retrieve our measurements. Initially, we provide information about our dataset and the ways we

[*]Produces the permission block, and copyright information
[†]The full version of the author's guide is available as `acmart.pdf` document
[‡]Dr. Trovato insisted his name be first.
[§]The secretary disavows any knowledge of this author's actions.
[¶]The secretary disavows any knowledge of this author's actions.
[1]Although in the physical sense energy cannot be consumed, we will use the terms energy "consumption", "requirement", and "usage" to refer to the conversion of electrical energy by ICT equipment into thermal energy dissipation to the environment. Correspondingly, we will use the term energy "savings", "reduction", "efficiency", and "optimization" to refer to reduced consumption

[2]http://smarter2030.gesi.org/downloads.php
[3]https://developer.android.com/ndk/index.html

decided to select our tasks and refine it. Furthermore, we explain the setup up of our experimental platform, the additional hardware tools, and the software tools used to conduct this research. Moreover, we argue on the selected tasks and programming languages we used and the way re refined our dataset.

3.1 Dataset

In the context of study, we used Rosetta Code,[4] a publicly available programming chromatography site that offers 851 tasks, 230 draft tasks, and a collection of 658 different programming languages. In general, not all of (and cannot) tasks are implemented in all languages. We found and downloaded a git hub repository[5] which contains all the currently implemented tasks found on Rosetta Code website.

For selecting the highly used programming languages, we made use of tiobe,[6] a software quality company. By making use of a formula[7], 25 of the highest ranked search engines (according to Alexa),[8] and a number of requirements enlisted for programming languages, tiobe provides a search query for index rating of the most popular programming languages around the web for each month. Initially, we decided of choosing the top 15 programming languages as enlisted for June 2017. From the current list, we excluded programming languages such as Delphi and Assembly. In contrast, we included Rust in our dataset which is a memory safe programming language and is gaining popularity in the web. Therefore, we ended up with 14 programming languages as illustrated in Table 2.

In terms of selecting tasks, we developed a shell script (more details in Subsection 3.2.2) to identify which of the 851 tasks offers the most implementations for the programming languages of our selection. To refine further our dataset we used the following steps:

- Some of the tasks offers more that one implementation for the same programming languages. Thus, we had to drive manually through each directory and remove them until we have only one that is consistent with the other implementation. For example, palindrome in recursion we removed the iterative implementations.
- The Java file's names were different from the public class names which results to compilation error if not change accordingly.
- Some of the implementations didn't have main classes, nor the same data with other tasks. Therefore, we change the source code to offer consistency.
- Some of the tasks are relatively small and may finish faster than 1 second which makes it impossible for our power analyzer to capture those results. Therefore, we added all the selected tasks in a loop to iterate them a million of times.

3.2 Hardware and Software components

3.2.1 Hardware Components. The physical tools composed mainly from a portable personal computer and a real-time electricity usage monitoring tool. Our system's specifications are depicted in Table 1. The real-time power usage tools we used is the Watts Up Pro (WUP).[9]

In general, there are two venues for retrieving energy consumption from a computer-based system: on one hand, by indirect energy measurements through estimation models or performance counters, core component of software monitoring tools, on the other hand, via direct measurement, hardware power analyzers and sensors. However, each approach has it own pitfalls such as: coarse-grained measurements for the whole systems' energy consumption and low sampling rate for direct measurements case and inaccuracy, lack of interoperability, and additional system overhead while using indirect measurements. Therefore, in our research we decided to retrieve our energy consumption measurements using direct approach such as WUP since our tasks are relatively short in terms of source-code lines.

In regards to WUP, it offers accuracy of ±1.5% and as maximum sampling rate of 1 second. In order to retrieve power-related measurements from the WUP we used a software available in a Git repository.[10] This software helped us retrieve measurements such as timestamps, watts, volts, amps, etc. through a mini USB interface after we integrated its code in our script that runs all the tasks. In order to avoid additional overhead in our measurements, we used a Raspberry Pi[11] in order to retrieve power consumption from our test-bed.

3.2.2 Software Components. To extract data, manage, and use our Rosetta Code Repository, we developed a number of shell scripts as enlisted below and are publicly available on our Git repository.[12]

- **script.cleanAll**, removes the current instance of Tasks in the current working directory and copy the new one found from in the parent directory.
- **script.fromUpperToLower**, changes the current instance of Tasks directory tree's letters from upper to lower case to offer consistency for our scripts.
- **script.findCommonTasksInLanguages**, provides a list of tasks with the amount of programming language implementations.
- **script.createNewDataSet**, filters the Rosetta Code current dataset and removes programming languages and tasks not added as command line arguments.
- **script.compileTasks**, compiles all tasks found under the Tasks' directory and produces error reports in a task fails to compile.
- **script.executeTasksRemotely**, executes all the tasks' implementations found in under Tasks directory. Moreover, it sends command to WUP, retrieves measurements and stores then on remote host, through *ssh*, in order to start retrieving measurements for each test case.
- **script.plotGraphs**, after retrieving our data we use this script to plot our graphs.

Note that most of the scripts offer the *–help* option that shows a list of available command line arguments in order use our scripts.

[4]http://rosettacode.org/wiki/Rosetta_Code
[5]https://github.com/acmeism/RosettaCodeData
[6]https://www.tiobe.com/tiobe-index/
[7]https://www.tiobe.com/tiobe-index/programming-languages-definition/
[8]http://www.alexa.com/
[9]https://www.wattsupmeters.com/secure/products.php?pn=0
[10]https://github.com/pyrovski/watts-up
[11]https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[12]https://github.com/stefanos1316/Rosetta-Code-Research

Table 1: System hardware and software specifications

	Description
Hardware	HP EliteBook 840 G3, Intel Core i7-6500U (2 physical cores of 2.5 GHz), 8 GB DDR4 memory, 256 GB SSD hard disk, Raspberry Pi Model 3b, 4ÃŮ ARM A53 1.2GHz, 2 GB LPDDR2 memory, 64 GB SD
Operating System	Fedora 25 kernel version 4.11.5-200, Raspbian
Software	WUP software (retrieving measurements from the device), bash script, Java, feedGnuplot

Table 2: Programming Languages, Compilers and Interpreters

	Programming Languages	Compilers and Interpreters version
Compiled	C, C++	gcc version 6.3.1 20161221-(Red Hat 6.3.1-1) (GCC)
	Go	go version go1.7.5
	Rust	rustc version 1.18.0
	VB.NET	vbnc version 0.0.0.5943[a]
Semi-Compiled	C#	mono version 4.4.2.0 (mics)[b]
	Java	javac version 1.8.0_131
Interpreted	JavaScript	node version 6.10.3
	Perl	perl version 5.24.1
	Php	php version 7.0.19
	Python	python version 2.7.13
	R	Rscript version 3.3.3
	Ruby	ruby version 2.3.3p222
	Swift	swift version 3.0.2[c]

[a] http://www.mono-project.com/docs/about-mono/languages/visualbasic/
[b] https://www.codetuts.tech/compile-c-sharp-command-line/
[c] https://github.com/FedoraSwift/fedora-swift2/releases/tag/v0.0.2

Moreover, users are suggested not to change the folders name or locations since most of our scripts are making use of them.

In addition, we used NTP[13] to synchronize both systems' clock. For plotting our graphs we used FeedGnuplot,[14] an open-source general purpose pipe-oriented plotting tool. To use our script for plotting graphs, someone has to provide a columned file with energy related measurements.

3.3 Retrieving Energy Measurements

As an initial step for our experiment, we shut down background process, as suggested by Hindle[...], found in modern OSs such as disk defragmentation, virus scanning, CRON jobs, automatic updates, disk indexing, document indexing, RSS feed updates, etc. to shorten some noise in our measurements. In addition, we set the network connection to flight mode and also reduced the brightness of the screen. By making the following steps we reduced our platform's idle power usage from 8,6 watts to 5.8 watts. We estimated that after an OS (Operating System) is launched it's necessary to wait for a short time to reach a *stable condition.*

After reaching *stable condition*, we launched our main script *i.e.,* **script.executeTasks**, that executes all the tasks implemented in different programming languages. While doing so, it retrieves power consumption and run-time performance measurements from WUP and through command time[15] and stores them in timestamped directories which we are using later to plot our results. Between each executing of a task, we added a sleep[16] period of three minutes. The time gap purpose is to ensure our experimental platform reached a stable condition,*e.g.,* the CPU is cooled down and the fan is no longer consuming more power, to avoid unnecessary noise in our measurements.

4 RESULTS AND DISCUSSION

Graphs for programming languages using compilers and interpreters

5 THREATS OF VALIDITY

6 CONCLUSION AND FUTURE WORK

Two Venues, Academic and Industrial. She under the curtain and shed light... Compare PL in different CPU architectures such as ARM and AMD. More tasks and different configuration and optimization flags. Collect resource usage and system calls to identify relationship among them.

Future plans, micro-services proposition of tasks or programming languages after identifying the reasons

REFERENCES

[1] Eugenio Capra, Chiara Francalanci, and Sandra A. Slaughter. 2012. Is Software "Green"? Application Development Environments and Energy Efficiency in Open Source Applications. *Inf. Softw. Technol.* 54 (Jan. 2012), 60–71.

[2] K. Eder. 2013. Energy transparency from hardware to software. In *2013 Third Berkeley Symposium on Energy Efficient Electronic Systems (E3S).* 1–2. https://doi.org/10.1109/E3S.2013.6705855

[3] M.A. Ferreira, E. Hoekstra, B. Merkus, B. Visser, and J. Visser. 2013. Seflab: A lab for measuring software energy footprints. In *2013 2nd International Workshop on Green and Sustainable Software (GREENS).* 30–37.

[4] Erol Gelenbe and Yves Caseau. 2015. The Impact of Information Technology on Energy Consumption and Carbon Emissions. *Ubiquity* 2015 (June 2015), 1:1–1:15. https://doi.org/10.1145/2755977

[5] Samir Hasan, Zachary King, Munawar Hafiz, Mohammed Sayagh, Bram Adams, and Abram Hindle. 2016. Energy Profiles of Java Collections Classes. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16).* ACM, New York, NY, USA, 225–236.

[6] C. Sahin, F. Cayci, I. L. M. GutiÃľrrez, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh. 2012. Initial explorations on design pattern energy usage. In *2012 First International Workshop on Green and Sustainable Software (GREENS).* 55–61. https://doi.org/10.1109/GREENS.2012.6224257

[7] Ward Van Heddeghem, Sofie Lambert, Bart Lannoo, Didier Colle, Mario Pickavet, and Piet Demeester. 2014. Trends in worldwide ICT electricity consumption from 2007 to 2012. *Computer Communications* 50 (Sept. 2014), 64–76. https://doi.org/10.1016/j.comcom.2014.02.008

[13] http://www.ntp.org/
[14] http://search.cpan.org/ dkogan/feedgnuplot-1.44/bin/feedgnuplot
[15] https://linux.die.net/man/1/time
[16] http://man7.org/linux/man-pages/man3/sleep.3.html