

Jolinar: Analysing the Energy Footprint of Software Applications (Demo)

Adel Noureddine, Syed Islam, Rabih Bashroush

▶ **To cite this version:**

Adel Noureddine, Syed Islam, Rabih Bashroush. Jolinar: Analysing the Energy Footprint of Software Applications (Demo). The International Symposium on Software Testing and Analysis, Jul 2016, Saarbrücken, Germany. pp.Pages 445-448, 10.1145/2931037.2948706 . hal-01348637

HAL Id: hal-01348637
https://hal.science/hal-01348637

Submitted on 29 Jul 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Jolinar: Analysing the Energy Footprint of Software Applications (Demo)

Adel Noureddine, Syed Islam, and Rabih Bashroush
School of Architecture, Computing and Engineering
University of East London
London, United Kingdom
a.noureddine@uel.ac.uk, syed.islam@uel.ac.uk, r.bashroush@qub.ac.uk

ABSTRACT

Monitoring energy consumption of applications is crucial for energy optimisation and improvements in software systems. With the recent emphasis on energy efficiency, it is vital that software engineers have an understanding of the energy consumed by the code they write. In this paper, we present `Jolinar`, a tool that bridges the gap between energy measurements and accessibility to software engineers and even end-users. The tool builds on top of recent energy models to provide an accurate, light and easy-to-use interface for energy measurements. The target audience of `Jolinar` is both software engineers and non-technical end-users who want to monitor their applications' energy footprint. We show that end-users can use `Jolinar`'s GUI to determine the energy consumed by the software they are using, and software engineers can use the tool to analyse energy consumption of systems to make energy-conscious decisions.

CCS Concepts

•**Hardware** → **Power estimation and optimization;** •**Software and its engineering** → *Empirical software validation;*

Keywords

Energy Footprint; Program Analysis

1. INTRODUCTION

With the proliferation of computer devices, the rate of energy consumption of Information and Communication Technologies (ICT) is rising at an alarming rate. It is estimated that ICT consumption will rise from 168 to 433 Gigawatts (7% to 14.5%) by 2020 [1]. Greenhouse Gas emissions (GHG) from ICT are also expected to double to 1430 MtCO2e within the same period [2], highlighting the need for more energy efficient hardware and designing sustainable software.

For developers to be able to enhance efficiency and optimise performance of software systems, they need to be able to measure energy consumption in order to identify bottlenecks, hotspots or energy smells in the code. Energy efficiency is an emerging software engineering quality that software architects consider as a major architectural concern in the next five years [3].

In addition to traditional software properties often cited to aid in program analysis and understanding [4], it is vital for software engineers to understand the energy consumed by their application and is an important part for program analysis and comprehension [5]. This is further highlighted by the recent treatment of energy consumption as a significant non-functional property in both industry and academia.

Accordingly, monitoring and optimising the energy consumption of software systems have gained considerable traction over the last few years. Researchers and practitioners are focusing on accurate models to estimate the energy consumption of software, which sometimes includes the use of complex formulas or additional hardware. However, limited research has been conducted on accessible tools to aid such effort and raise the awareness [6].

In this paper, we demonstrate `Jolinar`, our energy monitoring tool. `Jolinar` is an easy-to-use software tool that allows developers understand the energy consumed by their application. It is designed to be accessible by non-technical users and software engineers alike, providing an intuitive graphical user interface and a command-line tool for easy inclusion in external frameworks.

2. MOTIVATION & RELATED WORK

Current approaches for monitoring energy range from hardware devices to power models and tools [7]. Hardware-based solutions, such as power meter devices and dedicated integrated circuits or sensors, demand additional investment, complex installations, and, treat the entire system as a black box. Tools, such as pTop, PowerSpy, PowerAPI, JouleMeter or Energy Checker provide power and energy insights into software energy footprint but at the cost of limiting usability. For instance, some require a power meter in order to calibrate their models, or for runtime energy monitoring (PowerAPI, PowerSpy, JouleMeter). Others require modifications to applications' source code or patching parts of the operating system to effectively monitor software energy consumption (pTop, Energy Checker). While these modifications and additional investments might be accessible to researchers, this acts as a potential barrier for developers and end-users. This problem is further compounded by the

lack of access to physical hardware in modern software development cycles where deployment is moving to the cloud [8].

`Jolinar` addresses these challenges by providing a lightweight and easy to install and use energy monitoring tool. `Jolinar` requires no power meter or source code modifications enabling easy incorporation into any analysis toolkit. The visual information provided by `Jolinar` is pragmatic for precise energy data, while deploying familiar indicators and gauges that the non-energy experts can easily grasp.

3. JOLINAR DESIGN

`Jolinar` is based on our energy models [9]. These models were implemented in a system library tool called PowerAPI. PowerAPI was developed as an energy library that provides an API and is designed to scale to monitor multiple applications at the same time. It runs on personal computers and servers alike, but requires time consuming setup and an advanced level of technical and programming expertise to run, effectively making it difficult to use for end-users.

`Jolinar` bridges this gap between accurate energy estimations and usability. It monitors single applications at a time (with a limitation of one process per application), is easy to deploy and incorporate into an external framework and is also able to provide data in an easy to understand format. Our estimation models are based on hardware characteristics and software resource utilisation. The former are retrieved from publicly available OEM specifications, while resource utilisation is monitored at runtime with the help of metrics collected by the operating system. Our model uses this information to estimate the overall energy consumption of applications at runtime, as well as at the individual hardware component level. For example, we can estimate the total energy consumed by a particular application, and the breakdown per CPU, hard disk and memory components.

The estimation approach starts by monitoring hardware resource utilisation, such as the current frequency of the CPU or the number of bytes read or written to the hard disk. Next, OEM hardware specification data, along with resource utilisation information, are used by our estimation models to calculate the energy consumed by hardware components. Then, `Jolinar` monitors resource utilisation of the application, such as the CPU time or cycles used, and the number of bytes read or written. Finally, this information is used by the energy models to estimate the overall software energy consumption. Figure 1 summarises our energy estimation approach. Our CPU energy model [9] takes into consideration modern processors' characteristics, such as DVFS (Dynamic Voltage and Frequency Scaling), CPU's TDP (Thermal Design Power) and real-time processor frequency and voltage changes. The power consumption is calculated every 500 milliseconds, following power variation whenever frequency or voltage changes. Disk and memory energy models use a similar approach of capturing resources and estimating energy consumption. Disk energy model uses the number of bytes read and written by the application, while the memory mode uses the percentage of RAM memory occupied by the application.

Our energy models were validated with an error margin of around 3% on average [9]. However, we are aware that energy estimations may lose some accuracy on most recent releases of hardware. `Jolinar` is designed in a modular way where modules for energy models are independent of the user interface. Accordingly, when new energy models are designed to increase accuracy for latest hardware releases, `Jolinar` can easily be updated to reflect such changes.

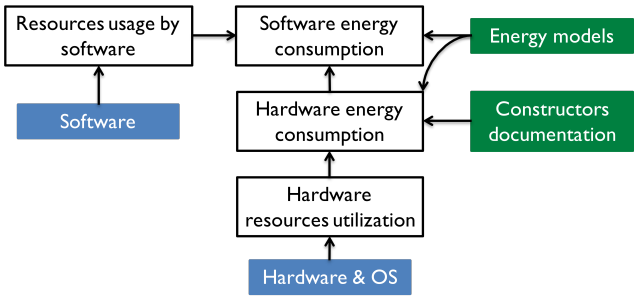


Figure 1: `Jolinar`'s energy estimation approach.

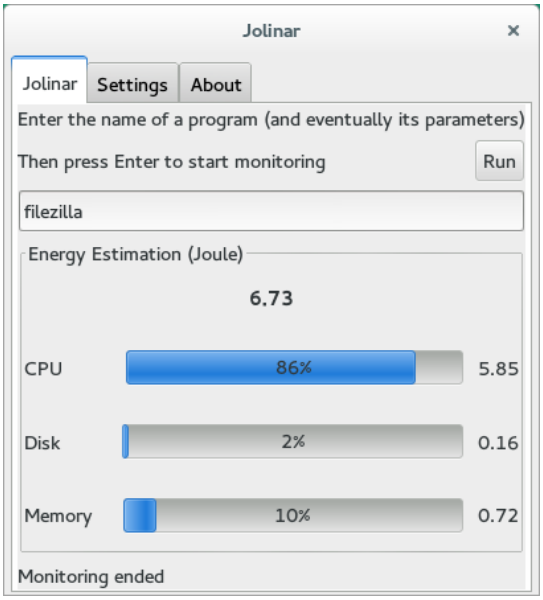


Figure 2: A successful energy monitoring by `Jolinar`.

4. END-USER USE CASE USING GUI

This section introduces the interface design and an end-user use case interaction of `Jolinar`. Figures 2, and 3, display an example of using `Jolinar` GUI to monitor the energy consumed by FileZilla, an FTP client on GNU/Linux. A typical use case where an end–user wants to understand the amount of energy consumed by usage of software.

`Jolinar` was designed to be an easy to install and use tool. The tool is distributed as *executable Jar*, *.deb* and *.rpm* packages where dependencies are automatically managed. As such, no compilation or command line instructions are required as the tool is packaged in *.deb* or *.rpm* packages.

The initial view of `Jolinar` is divided into two areas: the input box for choosing the application to monitor, and the visual output display, which shows the energy consumed by the application (Figure 2). Users simply need to indicate the name of the application to start monitoring. `Jolinar` launches the application and starts monitoring its energy consumption instantaneously. This presents the best trade-off between swiftness, monitoring and accessibility avoiding the need to deal with process ids. We chose to use a text

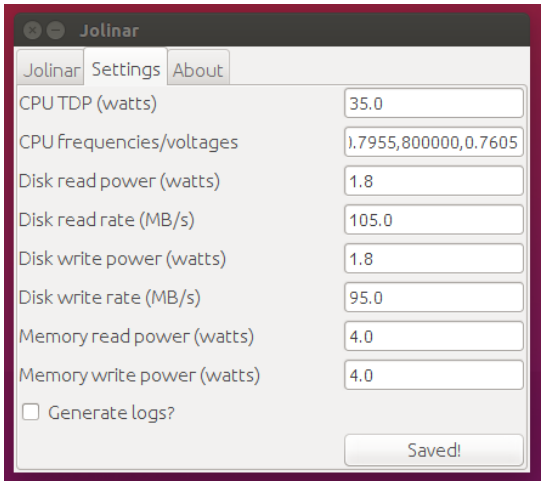


Figure 3: `Jolinar`'s settings view.

box to input the application name instead of a list to choose from, or permanently monitoring all applications, because this option presents the best trade-off between swiftness of monitoring and accessibility. Therefore, there is no need for software modification, memorising the application's process ID, or fiddling with a long list of monitored applications to find the one the user needs.

The second tab in `Jolinar`'s GUI window is where users can specify the settings and configuration information needed for the application and the energy models (Figure 3). Our models use hardware specification data that are usually provided by hardware OEMs, such as the TDP, frequency and voltage, or read/write power rates. An option to generate logs is also available for developers and will store runtime software power consumption (in Watts) each 500 milliseconds. When `Jolinar` launches the application and starts monitoring, it will do so in the background without impacting the performance of the application. Users can then use their software normally. On exit, `Jolinar` displays the energy estimations in both numerical values (in Joule and percentage) and using visual feedback. Energy estimations are broken into individual hardware components along with the total energy consumed by the application. This provides better understanding and awareness of which hardware component is responsible for the highest/lowest energy impact in the monitored application (which could help in identifying energy hotspots).

In the example of Figure 2, FileZilla consumed 6.73 Joules in a benchmark run consisting of sending and receiving 5 files to an FTP server, with most of that energy (5.85 Joules) being consumed by the processor (86% of the total energy of the application). Disk energy was around 2% due to reading files for FTP transfer, while the RAM memory consumed 10% of FileZilla's energy (0.72 Joules). This breakdown, both in Joules and in percentage, helps both software developers and end-users understand which hardware components are consuming the most energy. For developers, this will provide them with a first indication of the energy consumption of applications and help them focus their optimisation and improvement work to increase the energy efficiency of their software. Therefore, `Jolinar` is complementary, rather than a replacement, to tools such as `Jalen` [10] which monitors

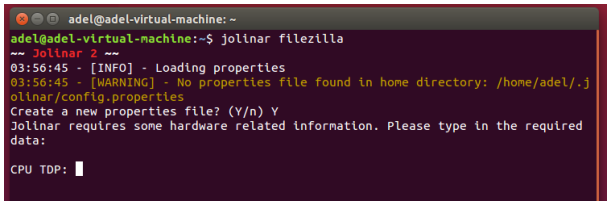


Figure 4: Command line version of `Jolinar`.

energy consumption of software source code (at the methods level). For end-users, the breakdown would allow them to better understand the energy footprint of each hardware component in their computer.

5. ANALYSIS USE CASE USING CLI

This section of the paper introduces the command line interface of `Jolinar` and shows its use in a use case where `Jolinar` is used to analyse libraries and executables, helping software engineers make energy conscious decision.

In addition to the graphical user interface, `Jolinar` provides a command-line tool (JCL) that is suited for large runs, scientific experiments, for headless applications in server configurations and incorporation into external frameworks. We designed the command line version to be as much intuitive and easy-to-use as the GUI one. Figure 4 displays the configuration wizard that is executed at the first run of `Jolinar` or when the configuration file is missing. The GUI and command-line versions both share the same *config.properties* file, giving users and developers two methods to update the settings (they can also edit the file directly).

The command-line tool offers simple commands to update settings or monitor energy with multiple options and flags. For instance, updating the TDP can be achieved by simply typing: *jolinar -tdp 20* to update its value to 20 Watts. Finally, monitoring an application follows a similar straightforward approach as the GUI, users only need to indicate the name of the application to monitor and `Jolinar` will execute, monitor and then output its energy consumption. JCL is suitable for inclusion in frameworks, such as GP that can consider energy consumption as fitness function to evolve programs.

We measure the energy consumed by two different algorithm implementations of calculating the digits of the number Pi. We measure the energy consumed by each algorithm to show the energy consumption for operating on the same set of inputs and computing the same set of data. We use y-cruncher [11], a software implementing multiple algorithms for calculating many constants - including Pi, to compare Chudnovsky's algorithm and Ramanujan's algorithm implementations on a GNU/Linux operating system. We compare the algorithms using a single thread and in a multi-threaded environment, for calculating digits of Pi ranging from 10 million to 100 million digits. We run the experiments on an HP G70 laptop with an Intel Dual T3400 processor at 2.16 GHz. The results in Figure 5 show that the Chudnovsky multi-threaded implementation (*Chudnovsky-MT*) consumes less than half the energy compared to Ramanujan's single-threaded one (*Ramanujan-Single*). More precisely, across all experiments, the multi-threaded version on average consumes 38% less energy compared to the single-

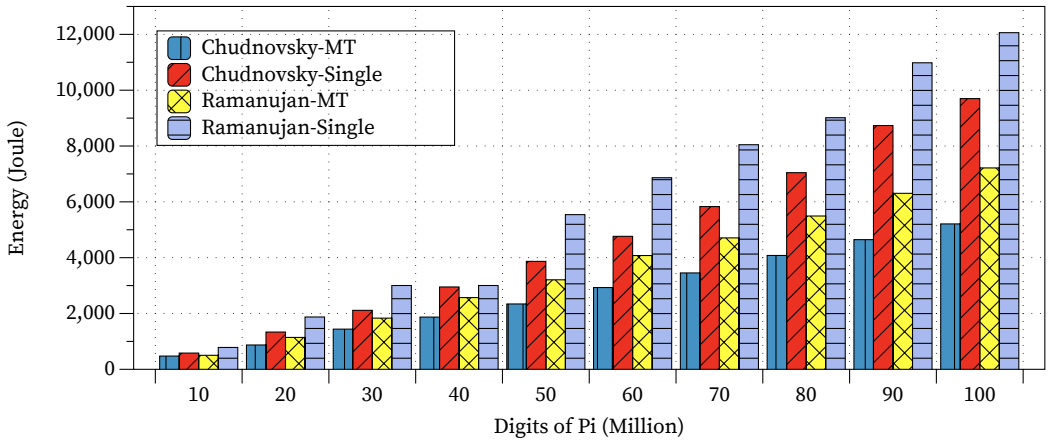


Figure 5: Energy consumption of Pi algorithms

threaded implementation. In addition, Ramanujan's algorithm has a 32% energy overhead, on average, compared to Chudnovsky's algorithm.

In cases where a software engineer is deciding whether to use an implementation of a Pi algorithm as a library feature, `Jolinar` would help them make an energy conscious decision thereby reducing the carbon footprint of the software. Profiling and analysis of energy consumption such as this can help developers identify the features that are consuming more energy and use additional information (such as importance and likely frequency of use) to decide on where to focus optimisation efforts (*e.g.* optimise code or dedicate energy efficient resources to frequently used features). Although software energy consumption varies from one platform to another, we identified in [10] that energy distribution within source code remains mostly stable with a variance of less than 4%.

6. CONCLUSIONS

In this paper, we present `Jolinar`, a software tool that can estimate the energy consumption of applications in an accurate, lightweight and accessible way to researchers, practitioners and non-technical end-users alike. With minimal setup and an automated approach, users only need to choose an application to monitor and `Jolinar` will provide detailed energy measurements with minimal effort. `Jolinar` is available as a free open-source software for GNU/Linux systems[1], and can monitor the energy consumption of any application.

We show how `Jolinar` can be used by end-users to identify energy consumption of software applications, as well as, aid software engineers profile energy usage and make energy-conscious decisions. We plan to improve the usability and the relevance of provided data with additional features, such as autotype and auto-search for applications to monitor; allow `Jolinar` to monitor already started applications; monitor mulitple process spwaned by an application; and provide energy consumption in monetary units.

7. REFERENCES

[1] W. Vereecken, W. Van Heddeghem, D. Colle, M. Pickavet, and P. Demeester. Overall ICT footprint and green communication technologies. In *Communications, Control and Signal Processing (ISCCSP), 2010 4th International Symposium on*, pages 1–6, March 2010.

[2] Molly Webb. *SMART 2020: enabling the low carbon economy in the information age, a report by The Climate Group on behalf of the Global eSustainability Initiative (GeSI)*. GeSI, 2008.

[3] R. Bashroush, E. Woods, and A. Noureddine. Data center energy demand: What got us here won't get us there. *IEEE Software*, 33(2):18–21, Mar 2016.

[4] M. A. Storey. Theories, methods and tools in program comprehension: past, present and future. In *13th International Workshop on Program Comprehension*, May 2005.

[5] Syed Islam, Adel Noureddine, and Rabih Bashroush. Measuring energy footprint of software features. In *IEEE International Conference on Program Comprehension*, To appear: May 2016.

[6] Gustavo Pinto, Fernando Castor, and Yu David Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 22–31, New York, NY, USA, 2014. ACM.

[7] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. A review of energy measurement approaches. *SIGOPS Oper. Syst. Rev.*, 47(3):42–49, November 2013.

[8] Anthony I Wasserman. Software engineering issues for mobile application development. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 397–400. ACM, 2010.

[9] Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, and Lionel Seinturier. A preliminary study of the impact of software engineering on greenit. In *Proceedings of the First International Workshop on Green and Sustainable Software*, Piscataway, NJ, USA, 2012. IEEE Press.

[10] Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. Monitoring energy hotspots in software. *Automated Software Engineering*, 22(3):291–332, 2015.

[11] y-cruncher, a multi-threaded pi-program. http://www.numberworld.org/y-cruncher/.

[1] http://www.noureddine.org/research/jolinar