Name:Janani M
Reg.No:422221104017
College Code:4222
Team No:T4

MEASURE ENERGY CONSUMPTION

Abstract

Over the years, the task to reduce energy consumed by a system has been mainly assigned to computer hardware developers. This is mainly because it is believed that the hardware is the principal component that consumes more electrical energy. However, the software also plays a vital role in power usage. Hardware works hand in hand with software programs. It has become equally important to estimate the energy consumed as a whole using artificial intelligence-based approaches. Machine learning is presented as one of the scalable approaches toward efficiently and accurately estimating energy consumed in the software development domain.

This chapter describes the theory and practical implementation of estimating the power consumption of computer devices. Different energy consumption estimation and monitoring approaches are discussed. The evolution of the methods and devices used to measure energy consumed by computer devices and their components is discussed. The challenges faced by the research community in measuring and monitoring energy consumption at different stages of software development are presented. Lastly a machine learning-based approach used for estimating energy consumption in a computer system running on a Windows or Linux operating system is presented. The proposed approach tries to address some of the challenges faced by the software development and research community in data retrieval and estimation of energy consumption in a system. The approach uses information about memory, CPU, and RAM utilization metrics to estimate the overall energy consumption.

Introduction

Over the years, the task to reduce energy consumed by a system has been mainly assigned to computer hardware developers. This is mainly because it is believed that the

hardware is the principal component that consumes more electrical energy. However, the software also plays a vital role in power usage. Hardware works hand in hand with software programs. Gradually over recent years software engineers have been putting more effort in developing green software. As evidence has been presented over the years, it has become clear that computers and other IT infrastructure consume significant amounts of electricity, placing a heavy burden on our electric grids and contributing to greenhouse gas emissions. For this reason, the field of green software engineering has emerged.

Green software engineering is concerned more about climate science, software architecture and practices, electronic devices power consumption, hardware, and data center design. The main question for green software engineers is about the greenness of software and hardware under development. Green software encompass three main phases of the software life cycle: (1) software usage, (2) software design, and (3) software implementation. The main goal is to reduce the amount of energy utilized in each of the phases and have minimal negative impact on the environment.

The first step toward designing green AI and software is measuring the energy consumption and pinpointing the different software components that increase the energy consumption (i.e., abnormal behavior recognition). The second step is recommending ways that could help increase the greenness of the system without tampering with the software's overall quality or performance (i.e., accuracy). It is worth noting that the measurement and recommendation are to be scalable and automated. The first step is addressed in this chapter.

Measuring the energy consumed by a computer system is a challenging task. The challenge is not only because of the fast-increasing advancements in technology but because of the different levels of granularity at which the energy can be measured and the introduction of cloud computing services. Cloud computing has increased the complexity of estimating the greenness of a software product at different phases (i.e., implementation and usage). In a very simple and basic case, hardware power meters are the solution for measuring the overall energy consumed by a physical computer system since they can provide readings about the energy consumption instantly. However, this approach does not help in pinpointing the computer components that need optimization because of high energy consumption.

Over the years, researchers have proposed different approaches which are alternatives to power meters and to measure energy consumption at different levels of granularity during the software development process (i.e., Alsultanny 2018; Bekaroo et al. 2016; Conti et al. 2016a; Mahesri et al. 2004). Among the proposed approaches machine learning-based approaches have been one of the most outstanding. Machine learning (ML) has found its way into many domains and has been outperforming traditional solutions to problems in different life applications including health-care systems (Chen et al. 2017), cyber security (Kabir et al. 2018), manufacturing (Martıet al. 2015), fraud detection (Zhang et al. 2018b), software quality assurance (Sidhu et al. 2022), and numerous other fields (Chandola et al. 2009). In this chapter, the ML-based approach is presented as another better, cheaper, and simple approach to measure energy consumed by a computer system in the software development process.

Energy Measurements Methods

Over the years, researchers and engineers have developed hardware and software tools to measure power consumption. Figure 3.1 presents a taxonomy of these developed tools and software for power consumption measurement.

Physical power monitors are the most accurate tools to measure the energy consumed by any device connected to an electric socket. Power monitors are directly connected to the power source of the device and measure the actual power leveraged at any instant of time. Despite the precision of the approach to measure the energy consumed by a system, it becomes unfeasible when scaled up. In addition to inability to scale, power meters can be challenging to set up (i.e., when measuring the energy consumed by a smartphone). An example of a power monitor can be seen in Fig. 3.2 taken fromFootnote1 with modifications. Figure 3.3 presents a general approach to measure the energy consumed by a system as it executes a specific set of instructions. As a result, most power monitors over the years have been integrated to software of a specific hardware which is believed to be the main source of energy consumption (i.e., CPU).

In a computer system, the CPU and processors are one of the main components that consume more energy. Intel, one of the giant computer chip developers, developed an energy measuring and monitoring tool called Intel Power Gadget. Intel Power Gadget (McKay et al. 2019) is an energy monitoring tool for Intel processors. It supports Windows and macOS. Intel Power Gadget provides callable APIs to get energy consumption information from code. In one of the latest updates, multi-socket system support was added. This has served as a first step toward awareness on the amount of energy consumed by a computer system as a whole. The awareness is not only dedicated to system administrators but also to software developers/programmers. Programmers

can develop software knowing how much energy it is going to consume as it spends more time utilizing the CPU.

Intel is not the only computer hardware manufacturer to have made efforts in developing tools that measure and monitor energy consumed by hardware such as the CPU. Microsoft developed a tool called Joulemeter (Joulemeter 2010). Joulemeter (Goraczko 2010) is no longer available for public download. Joulemeter estimates the energy consumption of various computer hardware components such as CPU and I/O devices. This tool is less accurate when compared to a physical power meter such as the one presented in Fig. 3.2. One of the main features of Joulemeter is that it distinguishes system energy, software energy, and VM energy. Nvidia, one of the leading video card producers, has developed a tool that measures and monitors the energy consumed by the graphics processing unit (GPU) hardware.

Coding

```
linters-settings:
  dupl:
    threshold: 100
  goconst:
    min-len: 2
    min-occurrences: 3
  gocritic:
    enabled-tags:
      - diagnostic
      - experimental
      - opinionated
      - performance
      - style
    disabled-checks:
      - dupImport # https://github.com/go-critic/go-critic/issues/845
      - ifElseChain
      - octalLiteral
  goimports:
```

```
    local-prefixes: github.com/golangci/golangci-lint

  govet:
    check-shadowing: true
    settings:
      printf:
        funcs:
          - (github.com/golangci/golangci-lint/pkg/logutils.Log).Infof
          - (github.com/golangci/golangci-lint/pkg/logutils.Log).Warnf
          - (github.com/golangci/golangci-lint/pkg/logutils.Log).Errorf
          - (github.com/golangci/golangci-lint/pkg/logutils.Log).Fatalf
  lll:
    line-length: 140
  misspell:
    locale: US

linters:
  disable-all: true
  enable:
    - bodyclose
    # - depguard
    - dupl
    - errcheck
    - goconst
    - gocritic
    - gocyclo
    - gofmt
    - goimports
    - goprintffuncname
    - gosimple
```

```
  - govet
  - ineffassign
  - misspell
  - nakedret
  - noctx
  - nolintlint
  - staticcheck
  - stylecheck
  - typecheck
  - unconvert
  - unparam
  - whitespace
  - unused
  - asciicheck

# don't enable:
# - lll
# - dogsled
# - gochecknoinits
# - gomnd
# - unused

# TODO: try to enable
# - scopelint
# - gochecknoglobals
# - gocognit
# - godot
# - godox
# - goerr113
# - interfacer
```

```
  # - maligned
  # - nestif
  # - prealloc
  # - testpackage
  # - revive
  # - wsl
  # - funlen

run:
  timeout: 5m

issues:
  exclude-rules:
    - linters:
        - typecheck
      text: "could not import C"
    - linters:
        - dupl
      text: ".*pkg/collector/metric/types/types_test.go.*" # false positive, https://github.com/mibk/dupl/issues/20
```

Software Tools

Software tools include software programs that estimate the system components' or the whole system energy consumption. Usually the estimation of energy consumed by a system is based on the system profile and the workload. Most approaches use mathematical models as bases for energy consumption. Over the years, researchers have developed mathematical models that determine the absolute level of power, which is necessary to transfer 1 Bit of Random Access Memory (RAM) from 0 to 1. Figure 3.4 presents a simple example of how the majority of software tools estimate energy consumption based on mathematical models.

The RAM is one of the power-sensitive components of a computer system. The developed mathematical models work hand in hand with energy-efficient hardware developers. The mathematical models are used to evaluate the greenness of developed hardware before being deployed into production. Multiple software tools to estimate energy consumption have been developed over the years, and some of them are operating system specific.

One of the tools dedicated to estimate energy consumption on Linux operating systems is called PowerTop (PowerTop n.d.). PowerTop is an open-source Linux tool for diagnosing issues with power consumption of computers. The main features of PowerTop are visualizing and presenting information about the energy consumed by each process running in the system. One drawback of PowerTop is that it relies on other external measurements of power.

Similar to PowerTop, McPAT (Li et al. 2009) is a framework for designing a processor. The main difference when compared to PowerTop is that it can model multicore and many core processors with a comprehensive structure. McPAT models not only performance metrics, but also energy metrics, based on analytical power models.

Most energy monitoring tools rely on data from the app/process while it is running. Usually, the main parameter is the CPU load. Often it runs as a service. pTop (Do et al. 2009) is a tool that provides information about energy consumption for the process. It uses data from the parts of the system that use the largest amount of energy: CPU, memory, NIC, and disk. One of the main advantages of pTop is the possibility to help developers create energy-optimized applications, and it provides energy-aware application programming interfaces.

Nadine Amsel and Bill Tomlinson created Green tracker (Amsel et al. 2010)—a tool that estimates the energy consumption of applications that is divided into groups. Users define groups themselves. At first, Green tracker uses benchmarking tests to get CPU statistics. Next, it saves information about time, process, and CPU usage at intervals; based on this data, it creates average CPU utilization for an application. Green tracker can be used to determine the application with the lowest power consumption. For example, Firefox 3.0.10 is better than Safari 4.0.3 in that aspect.

Brown et al. (2006) developed a battery life measurement toolkit for making reliable battery life measurements on Linux. This toolkit consists of a test framework and a number of example workloads (Idle, Reader, Office, DVD Player, SW Developer, and 3D-Gamer). The battery life measurement approach here uses a wall clock measurement from full charge until the battery is depleted.

Tools of this type collect information from all processes of the system, but because of this, it is hard for them to distinguish the applications with more than one process.

Tools of this group are created based on the power model of a specific processor. They count energy consumption for each processor instruction or group of processor instructions. This information can be used by software developers to optimize their application's power usage.

The first attempt to model such a power cost was described by Tiwari et al. (1994). The authors evaluated the energy cost of every instruction and took into account inter-instruction effects: circuit state, stalls, and cache misses. They found the base cost by looping in one instruction. Then researchers used a similar approach to sequences of instruction to obtain information about the circuit state effect. The next step was the experimental determination of the cost of the stall and the cache miss. In the end, they united all their results and created a framework that can estimate the energy consumption of a program.

Tiwari et al. (1996) continued to study this field and described two techniques for measurements: Board-Based Measurements and Tester-Based Measurements. They repeated the previous approach to three new processors and came up with the following observations: memory access is much more expensive than registry access, reordering instructions can reduce power consumption, each processor has its features, and developers can use them to create energy-efficient software.

Mazouz et al. (2017) presented a method that corresponds to both architecture simulators and program analysis groups, and its purpose is to estimate the energy consumption of multicore processors. The resulting energy is the sum of static and dynamic power. The static power is the power of core and uncore components: AlU, L1, L2, L3 cache, etc. Dynamic power is the energy consumed by instructions. Therefore, this method can help developers to correlate code and power usage.

Conti et al. (2016b) developed MTPlug, the framework for identifying laptop users from their energy traces. The framework is based on machine learning models for data preprocessing, segmentation, and feature extraction; however, the raw data comes from specific wall-socket smartmeters, which significantly reduce the practical implementation of this approach.

Li et al. (2003) proposed a method for estimating OS runtime energy consumption. The routine-level OS power model is a model where the average power of OS routines is known, and if we measure the execution time for each routine, then we can compute overall energy consumption.

Hybrid Tools

Hybrid methods create a collaboration between more than one measurement method (i.e., both software tools and hardware power meters). Such a collaboration assists to obtain more precise power readings that can be associated with software components.

E-Surgeon is a composition of PowerAPI and Jalen at a finer grain. Power consumption in e-Surgeon is estimated based on the program's methods and real-time execution. e-Surgeon collects information regarding CPU performance counters (e.g., time and network traffic) through both the operating system (via PowerAPI) and bytecode instrumentation or statistical sampling (via Jalen). e-Surgeon aggregates energy consumption collected by PowerAPI for multiple processes into one result.

PowerScope maintains a digital multimeter to profile and sample software energy consumption. Such a multimeter is controlled by another computer that collects all power readings. PowerScope maps energy consumption to the program structure and procedures. Power readings are collected at runtime but not analyzed until the program terminates. PowerScope is free of any profiling overhead since it performs a statistical sampling of the power consumption and the system activity over the collected measurements.

The Challenges of Estimating the Consumed Energy in Software Development

As the number of connected devices increases year by year worldwide, evaluating the energy efficiency of each device is becoming an ever more important and critical task. Understanding the energy fingerprint of computer devices is a step toward green IT and saving the planet. Being aware of the greenness of a software mainly in the software development domain contributes to designing energy-efficient software. The amount of energy consumed while developing a software product can be measured at different levels of granularity and software development. Measuring energy cost for developing a software product at different levels of granularity does not come at a low cost in terms of complexity.

Measuring energy consumed by any electrical appliance has been a challenging and complicated task on its own. The estimation and monitoring get more complicated when it comes to measuring the energy consumption of different components of a connection to other computers or devices. Understanding the root of the challenges is key in designing a framework to monitor and measure energy consumption in the software domain.

The first challenge is related to the metric that are is measured: power, which is a variable sinusoidal voltage. Accurately measuring the amount of energy consumed by a computer device is complex since most computer devices by their nature operate in a pulse signal mode when the current is not sinusoidal. The second problem is related to the accurate collection of energy consumption metrics in a noninvasive manner. Without the input of the device specifications, it is nearly impossible for the mathematical models (as presented in Fig. 3.4) to estimate the overall energy consumption. The third challenge is related to the advancements in technology and ever-increasing number of connected devices (IoT). In a case where part of the software uses cloud services, it is nearly impossible to measure the energy consumed by the cloud computing service such as the server. So calculating the exact amount of energy consumed by the software product that is under development gets more complex.

Machine Learning-Based Approach for Energy Consumption Measurement

This section presents a case study that proposes a machine learning (ML)-based approach to estimate the energy consumed by a system based on utilization of system core devices such as CPU, RAM, and virtual memory. In addition to the ML approach, it presents a use case of a noninvasive data collection framework.

Methodology

The proposed methodology for estimating energy consumed by a system in a noninvasive manner is presented in Fig. 3.5. This approach is based on machine learning and is implemented based on a framework called Innometrics. Details about each machine learning approach are presented in the following subsections.

Data Collection

The data collection and labeling task is one of the most challenging tasks in machine learning. Based on the nature of the problem that is to be solved in energy consumption estimation and monitoring, it is highly recommended that the approach be noninvasive and accurate. Innometrics (Ciancarini et al. 2020) provides the noninvasive feature of the proposed approach. Innometrics is available for three main operating systems (i.e., Windows, Mac OS, and Linux) in terms of data collectors deployed on the user devices. The data collectors retrieve data related to general information about the processes executed on the machine and extract their resource utilization such as CPU, RAM, and disk usage. For energy consumption monitoring purposes of the data about the resource utilization by each process, the data is sampled continuously in a uniform manner.

Data Preprocessing

Curating the data before learning the underlying data patterns is essential and crucial. To preprocess the data before parsing it to the machine learning model, we performed a couple of steps. Firstly, we removed all records with missing values. Secondly, the data records were aggregated according to the capture time using summation. We refrain from data scaling and data normalization since they do not make any significant impact on tree-based model performance. For feature selection, three features were selected to be used as model input, namely: (1) ratio of the process's resident set size to the physical memory on the machine, (2) virtual memory size of a process (vram), and (3) the central processing unit (CPU) utilization of a process. All the selected features are numerical.

Machine Learning Models

The proposed machine learning model here is an ensemble learning-based algorithm called CatBoost (Dorogush et al. 2018). The ML method was proposed and developed by Yandex researchers. CatBoost is an ensemble learning algorithm which is based on decision trees. The model hyper-parameters were tuned. The ability of the

model to be trained on accelerated hardware such as a graphics processing unit (GPU) made it easy to conduct a hyper-parameters search.

After a machine learning model is trained to recognize underlying data patterns and estimate the target (i.e., energy consumed by a system) given an input, it is important to evaluate the performance. To evaluate the ML model, three popular standard metrics are used: mean squared error (MSE), mean absolute error (MAE), and coefficient of determination (R 2). The calculation of the metrics is as follows:

$$\displaystyle \begin{aligned} MSE = \frac{1}{n}\sum_{t=1}^{n}\left (y_{i} - \hat{y}_{i} \right)^2 \end{aligned} $$ (3.1)

$$\displaystyle \begin{aligned} MAE = \left (\frac{1}{n} \right)\sum_{i=1}^{n}\left | y_{i} - \hat{y}_{i} \right | \end{aligned} $$ (3.2)

$$\displaystyle \begin{aligned} R^{2}=1-{ \sum_{t=1}^{n}\left (y_{i} - \hat{y}_{i} \right)^2 \over \sum_{t=1}^{n}\left (y_{i} - \bar{y}_{i} \right)^2} \end{aligned} $$ (3.3)

where \(\hat {y}\) is the predicted value, n is the total number of data samples, and \(\bar {y}\) is the mean value of y calculated by \(\frac {1}{n} \sum _{i=1}^{n} y_i\).

Conclusion

The presented results provide a starting point for the research community as it tries to accurately estimate the amount of energy consumed in software development leveraging the benefits of ML. However, there is still room for improvement in the proposed approach above. The challenges faced by the research community have been presented and the existing energy consumption measurement tools have also been discussed.