SOFTWARE AND
SUSTAINABILITY
S2 RESEARCH GROUP

Design and Development of Green Software

Vincenzo Stoico @ GreenLab

PostDoctoral Researcher

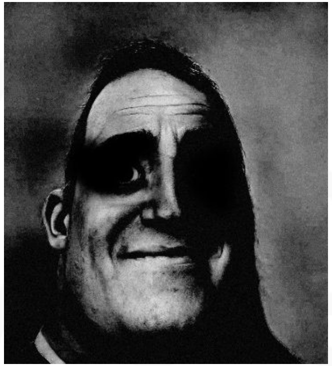
v.stoico@vu.nl

VRIJE UNIVERSITEIT AMSTERDAM

Introduction

Growth of ICT devices and services

Impact on People's Lives | **Energy Demand**

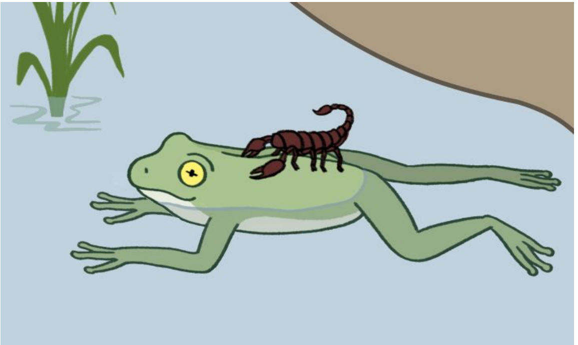


Belkhir et al. estimate that ICT devices will produce 14% of global CO_2 emissions by 2040 [1]

[1] Belkhir, L., Elmeligi, A.: *Assessing ICT global emissions footprint: Trends to 2040 & recommendations*, Journal of Cleaner Production, 2018

IMG: https://anacurbelol.com/PG-Illustrations

Introduction



HW power consumption **savings** (Frog)

Poor design decisions at the SW level (Scorpion)

"**Software-related CO2 emissions account for 4-5%** of global emissions. This is equivalent to the emissions of all aviation, shipping, and rail combined" [2]

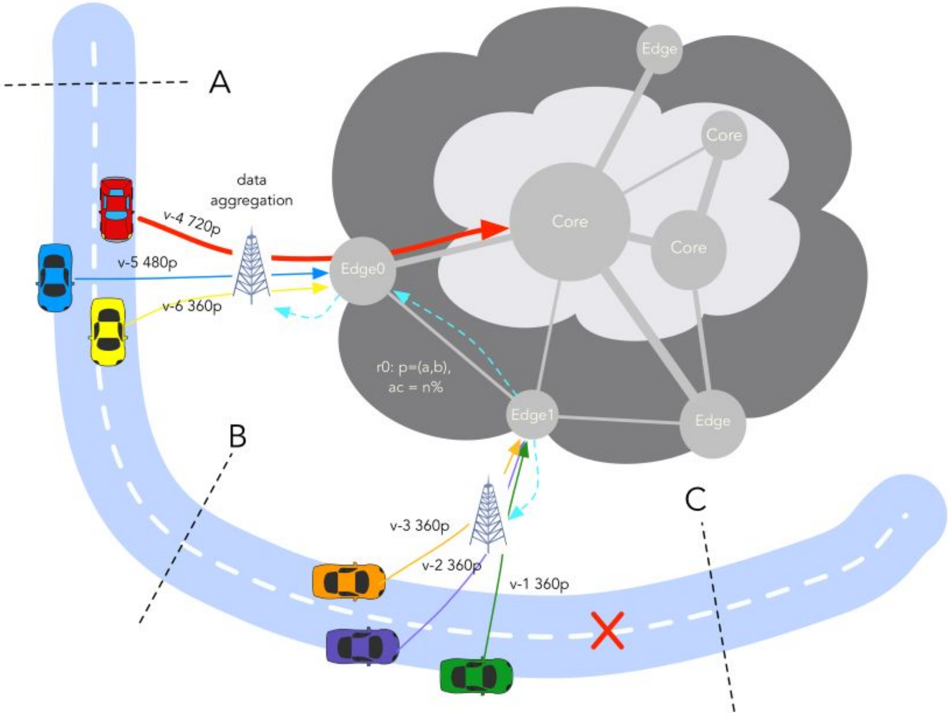
Techniques to reduce **SW energy consumption** are crucial to achieve Net Zero Goals

[2] Green Software Foundation, *2023 State of Green Software*, https://stateof.greensoftware.foundation/insights/software-emissions-are-equivalent-to-air-rail-shipping-combined/
IMG: https://anacurbelol.com/PG-Illustrations

An holistic view of software energy consumption

- **Optimizing** overall energy consumption is **complex**
- SoA offers **domain-specific energy models/techniques**, <u>none of them provides the overall picture</u>
- Identify **energy hotspots**
- Exploit **Modeling** and **Simulation**

***Inductive approach*:** we collect empirical evidence that we analyze



Y. Li et al., "End-to-end energy models for Edge Cloud-based IoT platforms: Application to data stream analysis in IoT" . Future Generation Computer Systems, vol. 87, pp. 667–678, Oct. 2018, doi: 10.1016/j.future.2017.12.048.

Green Architectural Tactics for the Cloud

tactics: "***design decisions*** that influence the achievement of **a quality attribute response**"

Example: Apply Edge Computing

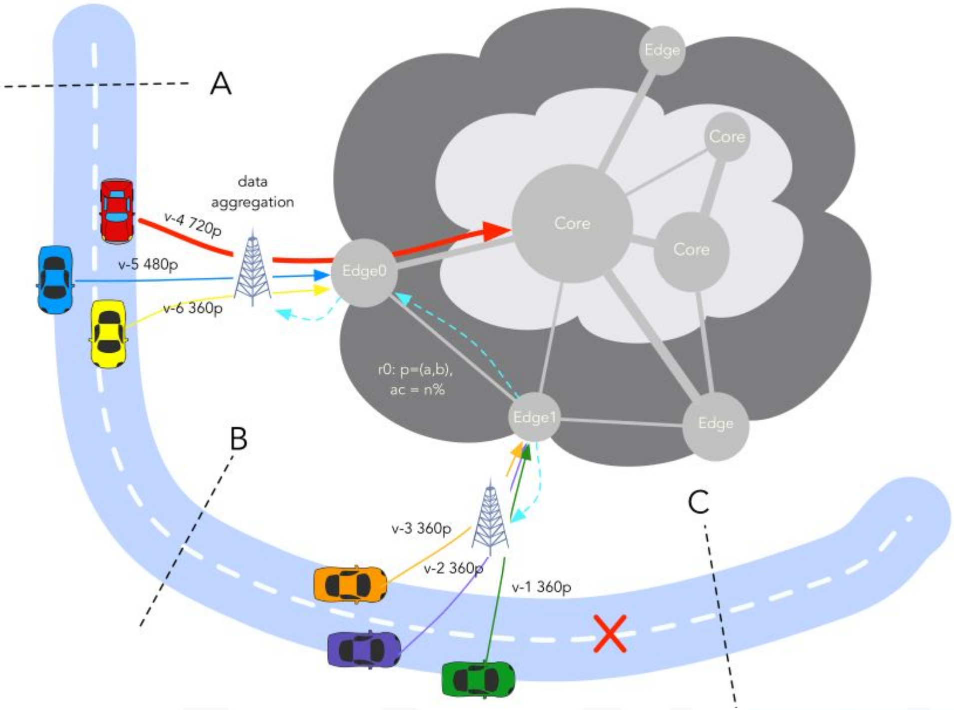
Real-Time Object Detection

QoS depends on connectivity

Edge Benefits:

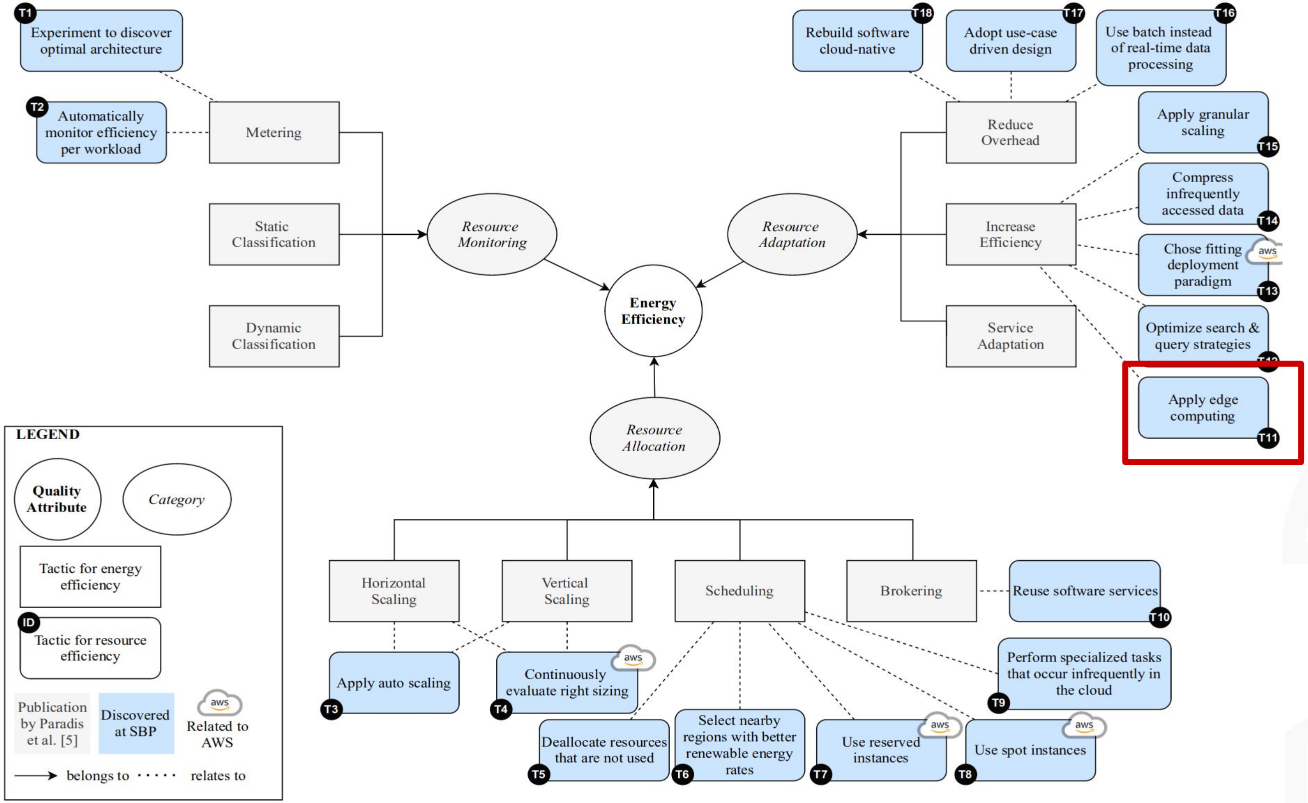
Reduced Latency

Energy Savings



Y. Li et al., "End-to-end energy models for Edge Cloud-based IoT platforms: Application to data stream analysis in IoT" . Future Generation Computer Systems
Procaccianti et. al, "Green Architectural Tactics for the Cloud", 2014 IEEE/IFIP Conference on Software Architecture, Sydney, Australia, 2014

Green Architectural Tactics for the Cloud



S. Vos, P. Lago, R. Verdecchia and I. Heitlager, "Architectural Tactics to Optimize Software for Energy Efficiency in the Public Cloud", 2022 International Conference on ICT for Sustainability (ICT4S), Plovdiv, Bulgaria, 2022

VU VRIJE UNIVERSITEIT AMSTERDAM

Outline

- Energy Efficiency Across **Programming Languages**
- Empirical Evaluation of **Two Best Practices** for Energy-Efficient Software Development

Measurement-Based

- Catalog of **Energy Patterns** for **Mobile** Applications

Data Mining

- An Approach Using Performance **Models** for Supporting Energy Analysis of Software Systems
- An independent assessment and improvement of the **Digital Environmental Footprint formulas**

Model-Based

In this lecture, you will find:

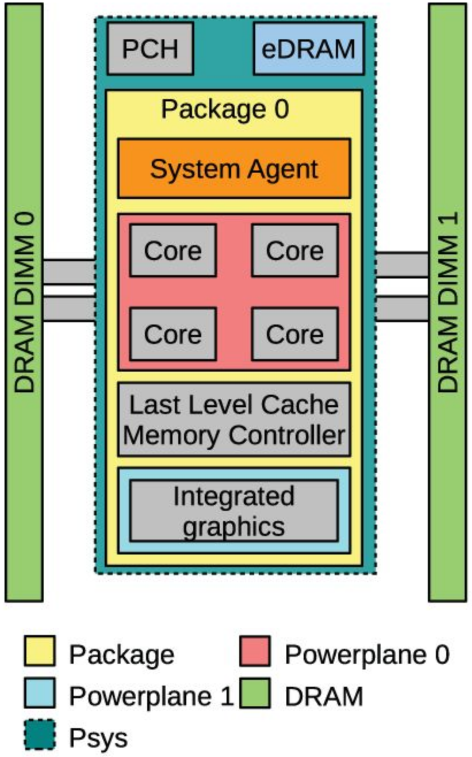
- **Tools** and **approaches** for **evaluating** SW energy consumption
- **Well-conducted** experiments

Running Average Power Limit (RAPL)

Interface provided by Intel and implemented on **modern** Intel/AMD processors

- **PKG**: The entire package
  - PP0: The cores.
  - PP1: An uncore device, usually the GPU (not available on all processor models.)
- **DRAM**: main memory (not available on all processor models.)

The following relationship holds: PP0 + PP1 <= PKG. DRAM is independent of the other three domains.



https://firefox-source-docs.mozilla.org/performance/power_profiling_overview.html

RAPL support

- Supported by Intel Processors since Intel **SandyBridge** Architecture **(2011)**
- Supported by AMD Processors since **AMD Family 17h** Processors **(2017)**
- **there isn't** any RAPL-like event for **ARM**
  - Use Power Monitor (e.g., INA219)
  - Estimations

RAPL-based Tools:

- Intel Power Gadget *(Windows/Mac)*
- Powerstat/Powertop/perf *(Linux)*
- Powermetrics (Mac)
- SmartWatts (Linux)

```
#define MSR_RAPL_POWER_UNIT      0x606

/*
 * Platform specific RAPL Domains.
 * Note that PP1 RAPL Domain is supported on 062A only
 * And DRAM RAPL Domain is supported on 062D only
 */
/* Package RAPL Domain */
#define MSR_PKG_RAPL_POWER_LIMIT    0x610
#define MSR_PKG_ENERGY_STATUS       0x611
#define MSR_PKG_PERF_STATUS      0x613
#define MSR_PKG_POWER_INFO       0x614
```

https://github.com/amd/amd_energy

RAPL support

- Supported by Intel Processors since Intel **SandyBridge** Architecture **(2011)**
- Supported by AMD Processors since **AMD Family 17h** Processors **(2017)**
- **there isn't** any RAPL-like event for **ARM**
  - Use Power Monitor (e.g., INA219)
  - Estimations

RAPL-based Tools:

- Intel Power Gadget *(Windows/Mac)*
- Powerstat/Powertop/perf *(Linux)*
- Powermetrics (Mac)
- SmartWatts (Linux)

Supported

```
vincenzo@GreenLab-STF:/sys/devices/platform$ sudo rdmsr 0x606
a0e03
vincenzo@GreenLab-STF:/sys/devices/platform$
```

Not Supported

```
(base) vincenzo@gl4:/sys/devices/platform$ sudo rdmsr 0x606
rdmsr: CPU 0 cannot read MSR 0x00000606
(base) vincenzo@gl4:/sys/devices/platform$
```

https://github.com/amd/amd_energy

Outline

- Energy Efficiency Across **Programming Languages**
- Empirical Evaluation of **Two Best Practices** for Energy-Efficient Software Development

Measurement-Based

- Catalog of **Energy Patterns** for **Mobile** Applications

Data Mining

- An Approach Using Performance **Models** for Supporting Energy Analysis of Software Systems
- An independent assessment and improvement of the **Digital Environmental Footprint formulas**

Model-Based

Energy Efficiency Across Programming Languages

Energy Efficiency across Programming Languages

How Do Energy, Time, and Memory Relate?

Rui Pereira
HASLab/INESC TEC
Universidade do Minho, Portugal
ruipereira@di.uminho.pt

Marco Couto
HASLab/INESC TEC
Universidade do Minho, Portugal
marco.l.couto@inesctec.pt

Francisco Ribeiro, Rui Rua
HASLab/INESC TEC
Universidade do Minho, Portugal
fribeiro@di.uminho.pt
rrua@di.uminho.pt

Jácome Cunha
NOVA LINCS, DI, FCT
Univ. Nova de Lisboa, Portugal
jacome@fct.unl.pt

João Paulo Fernandes
Release/LISP, CISUC
Universidade de Coimbra, Portugal
jpf@dei.uc.pt

João Saraiva
HASLab/INESC TEC
Universidade do Minho, Portugal
saraiva@di.uminho.pt

Abstract

This paper presents a study of the runtime, memory usage and energy consumption of twenty seven well-known software languages. We monitor the performance of such languages using ten different programming problems, expressed in each of the languages. Our results show interesting findings, such as, slower/faster languages consuming less/more energy, and how memory usage influences energy consumption. We show how to use our results to provide software engineers support to decide which language to use when energy efficiency is a concern.

CCS Concepts • **Software and its engineering → Software performance**; **General programming languages**;

Keywords Energy Efficiency, Programming Languages, Language Benchmarking, Green Software

ACM Reference Format:
Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, and João Saraiva. 2017. Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate?. In *Proceedings of 2017 ACM SIGPLAN International Conference on Software Language Engineering (SLE'17)*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3136014.3136031

1 Introduction

productivity - by incorporating advanced features in the language design, like for instance powerful modular and type systems - and at efficiently execute such software - by developing, for example, aggressive compiler optimizations. Indeed, most techniques were developed with the main goal of helping software developers in producing faster programs. In fact, in the last century *performance* in software languages was in almost all cases synonymous of *fast execution time* (embedded systems were probably the single exception).

In this century, this reality is quickly changing and software energy consumption is becoming a key concern for computer manufacturers, software language engineers, programmers, and even regular computer users. Nowadays, it is usual to see mobile phone users (which are powerful computers) avoiding using CPU intensive applications just to save battery/energy. While the concern on the computers' energy efficiency started by the hardware manufacturers, it quickly became a concern for software developers too [28]. In fact, this is a recent and intensive area of research where several techniques to analyze and optimize the energy co... sumption of software systems are being developed. S... techniques already provide knowledge on the ene... ciency of data structures [15, 27] and android lan... the energy impact of different programming pr... mobile [18, 22, 31] and desktop applicatio...

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

Energy Efficiency Across Programming Languages

Motivation:
Provide software engineers **support** to decide **which language** to use when energy **efficiency** is a concern

Method:
Profile **10 well-known problems** implemented in **27 programming languages**

Research Questions:

RQ1 Can we **compare** energy efficiency of SW languages?

RQ2 Is the **faster** language always the **most** energy efficient?

RQ3 How does **memory usage** relates to energy consumption?

RQ4 Can we **automatically decide** the **best** SW language considering execution time, energy consumption, memory?

Computer Language Benchmarks Game (CLBG)

CLBG is a **framework** for running, testing and comparing programming languages

Born in 00s for comparing scripting languages. Nowadays, it includes **13 problems** implemented in 28 programming languages

fannkuch-redux

source	secs	mem	gz	cpu secs
C++ g++ #6	3.23	10,936	1528	12.80
Rust #6	3.51	11,036	1253	13.93
C++ g++ #7	14.04	10,912	1150	14.04
Rust #4	7.21	10,932	1020	28.34

Benchmark	Description
n-body	Double precision N-body simulation
fannkuch-redux	Indexed access to tiny integer sequence
spectral-norm	Eigenvalue using the power method
mandelbrot	Generate Mandelbrot set portable bitmap file
pidigits	Streaming arbitrary precision arithmetic
regex-redux	Match DNA 8mers and substitute magic patterns
fasta	Generate and write random DNA sequences
k-nucleotide	Hashtable update and k-nucleotide strings
reverse-complement	Read DNA sequences, write their reverse-complement
binary-trees	Allocate, traverse and deallocate many binary trees
chameneos-redux	Symmetrical thread rendezvous requests
meteor-contest	Search for solutions to shape packing puzzle
thread-ring	Switch from thread to thread passing one token

Computer Language Benchmarks Game (CLBG), https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html, Accessed 09-09-2023

Experiment Design and Execution

- **Most efficient version** (i.e. fastest) version of the source code
- Replicated **the information** of the CLBG
- **Functional Correctness** Verification
- Each benchmark has been executed <u>10 times</u>
- **Peak Memory Usage** measured with using `/usr/bin/time -v command`

```
...
for (i = 0 ; i < N ; i++){
  time_before = getTime(...);
  //performs initial energy measurement
  rapl_before(...);

  //executes the program
  system(command);

  //computes the difference between
  //this measurement and the initial one
  rapl_after(...);
  time_elapsed = getTime(...) - time_before;
  ...
}
...
```

Figure: Measurement Framework

RQ2: Is Faster, Greener?

No, a faster language is **not always** the most energy efficient

- Energy (J) = Power (W) x Time (s)

Fastest and most *Energy Efficient* Languages:

- Compiled
- Imperative

87-88% of the energy consumption **derived from the CPU** and the remaining to the DRAM

fasta				
	Energy	Time	Ratio	Mb
(c) Rust $\Downarrow_9$	26.15	931	0.028	16
(c) Fortran $\downarrow_6$	27.62	1661	0.017	1
(c) C $\uparrow_1$ $\Downarrow_1$	27.64	973	0.028	3
(c) C++ $\uparrow_1$ $\Downarrow_2$	34.88	1164	0.030	4
(v) Java $\uparrow_1$ $\Downarrow_{12}$	35.86	1249	0.029	41
(c) Swift $\Downarrow_9$	37.06	1405	0.026	31
(c) Go $\downarrow_2$	40.45	1838	0.022	4
(c) Ada $\downarrow_2$ $\Uparrow_3$	40.45	2765	0.015	3
(c) Ocaml $\downarrow_2$ $\Downarrow_{15}$	40.78	3171	0.013	201
(c) Chapel $\uparrow_5$ $\Downarrow_{10}$	40.88	1379	0.030	53
(v) C# $\uparrow_4$ $\Downarrow_5$	45.35	1549	0.029	35
(i) Dart $\Downarrow_6$	63.61	4787	0.013	49
(i) JavaScript $\Downarrow_1$	64.84	5098	0.013	30
(c) Pascal $\downarrow_1$ $\Uparrow_{13}$	68.63	5478	0.013	0
(i) TypeScript $\downarrow_2$ $\Downarrow_{10}$	82.72	6909	0.012	271
(v) F# $\uparrow_2$ $\Uparrow_3$	93.11	5360	0.017	27
(v) Racket $\downarrow_1$ $\Uparrow_5$	120.90	8255	0.015	21
(c) Haskell $\uparrow_2$ $\Downarrow_8$	205.52	5728	0.036	446
(v) Lisp $\Downarrow_2$	231.49	15763	0.015	75
(i) Hack $\Downarrow_3$	237.70	17203	0.014	120
(i) Lua $\Uparrow_{18}$	347.37	24617	0.014	3
(i) PHP $\downarrow_1$ $\Uparrow_{13}$	430.73	29508	0.015	14
(v) Erlang $\uparrow_1$ $\Uparrow_{12}$	477.81	27852	0.017	18
(i) Ruby $\downarrow_1$ $\Uparrow_2$	852.30	61216	0.014	104
(i) JRuby $\uparrow_1$ $\Downarrow_2$	912.93	49509	0.018	705
(i) Python $\downarrow_1$ $\Uparrow_{18}$	1,061.41	74111	0.014	9
(i) Perl $\uparrow_1$ $\Uparrow_8$	2,684.33	61463	0.044	53

Energy Efficiency Across Programming Languages, https://sites.google.com/view/energy-efficiency-languages, Accessed 09-09-2023

RQ3: Memory Impact on Energy

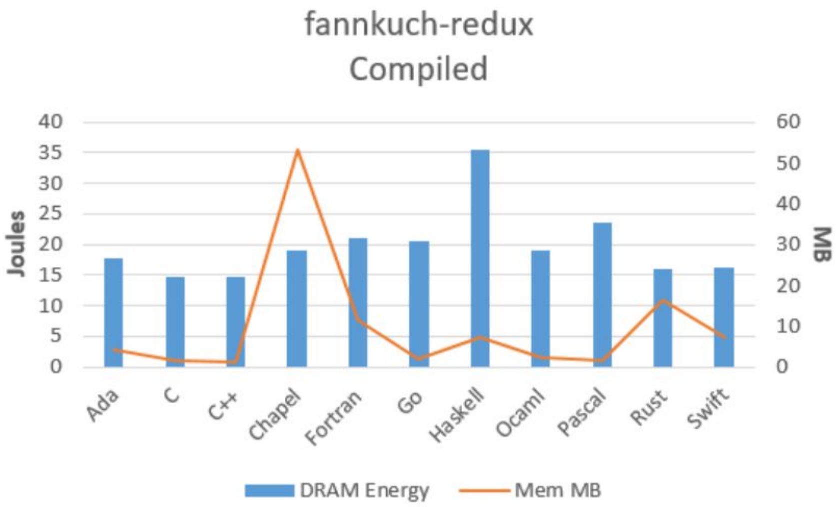
Peak memory usage: how memory is saved at a given point of the execution

Best Languages:

- Imperative
- Compiled

No correlation between DRAM energy consumption and peak memory usage

ToDo: correlation between energy consumption and *continuous memory usage*



RQ4: Energy vs Time vs Memory

Time & Memory	Energy & Time	Energy & Memory	Energy & Time & Memory
C • Pascal • Go	C	C • Pascal	C • Pascal • Go
Rust • C++ • Fortran	Rust	Rust • C++ • Fortran • Go	Rust • C++ • Fortran
Ada	C++	Ada	Ada
Java • Chapel • Lisp • Ocaml	Ada	Java • Chapel • Lisp	Java • Chapel • Lisp • Ocaml
Haskell • C#	Java	OCaml • Swift • Haskell	Swift • Haskell • C#
Swift • PHP	Pascal • Chapel	C# • PHP	Dart • F# • Racket • Hack • PHP
F# • Racket • Hack • Python	Lisp • Ocaml • Go	Dart • F# • Racket • Hack • Python	JavaScript • Ruby • Python
JavaScript • Ruby	Fortran • Haskell • C#	JavaScript • Ruby	TypeScript • Erlang
Dart • TypeScript • Erlang	Swift	TypeScript	Lua • JRuby • Perl
JRuby • Perl	Dart • F#	Erlang • Lua • Perl	
Lua	JavaScript	JRuby	
	Racket		
	TypeScript • Hack		
	PHP		
	Erlang		
	Lua • JRuby		
	Ruby		

Summary

- ***Compiled and Imperative*** programming language **perform better** and **more energy/memory efficient**

- It is not possible to find a programming language that **improves all three attributes**

- **CPU** seems consuming most of the **energy consumption**

- An evaluation of memory usage over time **is missing**

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

VU VRIJE UNIVERSITEIT AMSTERDAM

Empirical Evaluation of Two Best Practices for Energy-Efficient Software Development

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

ELSEVIER

Empirical evaluation of two best practices for energy-efficient software development

Giuseppe Procaccianti*, Héctor Fernández, Patricia Lago

VU University Amsterdam, De Boelelaan 1081a, 1081 HV, Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 12 July 2015
Revised 16 December 2015
Accepted 23 February 2016
Available online 4 March 2016

Keywords:
Software engineering
Best practices
Energy efficiency

ABSTRACT

Background. Energy efficiency is an increasingly important property of software. A larg
pirical studies have been conducted on the topic. However, current state-of-the-Art
empirically-validated guidelines for developing energy-efficient software.

Aim. This study aims at assessing the impact, in terms of energy savings, of best pract
software energy efficiency, elicited from previous work. By doing so, it identifies wh
affected by the practices and the possible trade-offs with energy consumption.

Method. We performed an empirical experiment in a controlled environment, where
different Green Software practices to two software applications, namely query optim
Server and usage of "sleep" instruction in the Apache web server. We then performed
the energy consumption at system-level and at resource-level, before and after applyin

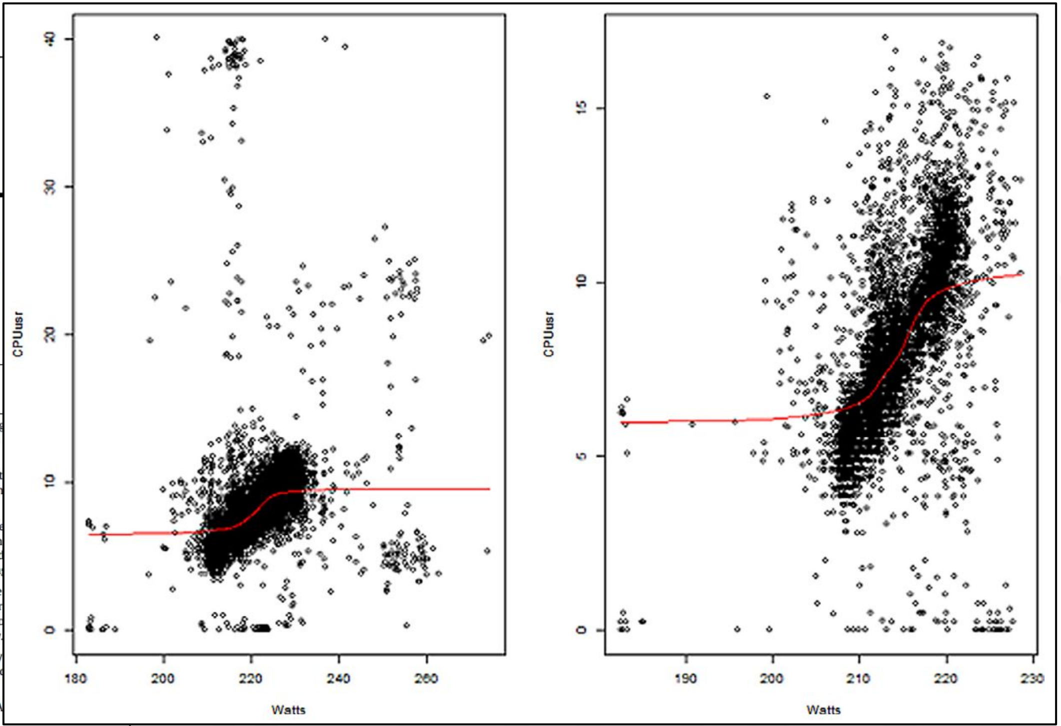
Results. Our results show that both practices are effective in improving software ene
ducing consumption up to 25%. We observe that after applying the practices, resour
energy-proportional i.e., increasing CPU usage increases energy consumption in an almo
also provide our reflections on empirical experimentation in software energy efficiency.

Conclusions. Our contribution shows that significant improvements in software energy
gained by applying best practices during design and development. Future work will be d
validate best practices, and to improve their reusability.

1. Introduction

The energy impact of software has been recognized as significant with respect to the overall energy consumption of its execution environment (Capra et al., 2012b; Procaccianti et al., 2012). Many researchers have been working on sophisticated software power models (Sinha and Chandrakasan, 2000; Kansal and Zhao, 2008) able to estimate and predict the energy consumption of software applications through different parameters. In spite of this ef-

To understand how software can impact on energy consumption, consider the following example[1]: after launch, the popular Youtube video of the "Gangnam Style" song reached a record amount of visualizations during the first year after its publicatio roughly 1.7 billion. The amount of energy used by Google to fer 1 MB across the Internet (as reported by the company website[2]) is 0.01 kWh (a rough average), and display 0.002 kWh (depending on the destination device). H ergy needed to stream and display the "Gangnam



VU VRIJE UNIVERSITEIT AMSTERDAM

Empirical Evaluation of Two Best Practices for Energy-Efficient Software Development

Motivation:
Current SoA does not provide **empirical evidence** of tactics for green software

Method:
Controlled Experiment in which **two practices** were empirically evaluated

Research Questions

RQ1: What is the **impact of each practice** in terms of energy consumption?

RQ2: Is the **relationship** between **resources and power consumption** affected by the application of each practice?

VU VRIJE UNIVERSITEIT AMSTERDAM

Experiment Design

Two Practices: (1) *Put application to sleep* and (2) *Use Efficient Query*

Quasi-Experiment:

Practices **manually** applied to two open-source SW applications:
Apache Web Server for (1) and MySQL Database Server for (2)

Dependent Variables:

1. Energy Consumption at **System-Level**
2. Energy Values of **Each Resource** (CPU, Disk, Network, Memory)

Independent Variables:

- *Fixed* Workload
- Absence/Application of a Green SW Practice (2 Treatments)
- *Fixed* Test machine (HW/SW)

VRIJE UNIVERSITEIT AMSTERDAM

Experiment Execution

10 executions for each practice

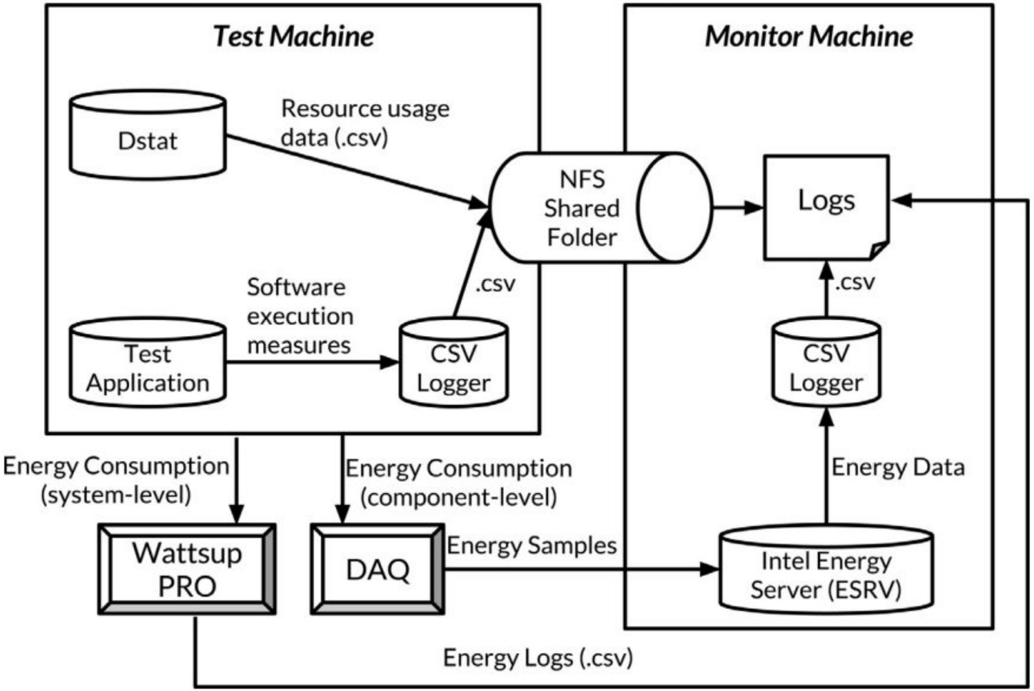


Figure: Experiment Setting

Experiment Execution

Practice 1: Use Efficient Queries:

- Database **populated** with the English Version of Wikipedia (30GB)
- **Query searching** for text fragments

```
SELECT SQL_NO_CACHE a.old_id
FROM text a, revision b
WHERE a.old_id = b.rev_text_id
ORDER BY a.old_id;
```

Figure: Query before applying the practice

```
SELECT SQL_NO_CACHE a.old_id
FROM text a, revision b
WHERE a.old_id = b.rev_text_id
```

Figure: Query after applying the practice

Practice 2: Put Application to Sleep

- sleep() while waiting for a HTTP Request
- Workload made of **5 million** requests with **max 50 concurrent requests** and a time limit of **5 min** (ab utility)

Efficient Query - Results

RQ1: What is the impact of each practice in terms of energy consumption?

- **Low decrease** in **Power Consumption** due to *performance optimization*

RQ2: Is the relationship between resources and power consumption affected by the application of each practice?

- **Direct Correlation** between **CPU and Disk Consumption**
- **After** applying the practice, the correlation I/O operations and Energy have **negative correlation** (CPU Inactive)

Put Application to Sleep - Results

RQ1: What is the impact of each practice in terms of energy consumption?

- Almost **no difference between Power and Energy Consumption Improvement** (correlation between performance and energy)

RQ2: Is the relationship between resources and power consumption affected by the application of each practice?

- Confirmed **Energy-Proportional** Behavior
- CPU not the main driver of energy consumption since Memory has **the same consumption**

Summary

- The paper confirms the **importance** of Green Software Tactics
  - **Significant Energy Reduction (25%)**
  - **Impact** on Resource Consumption
- Energy Consumption should be considered **a first-class design concern**

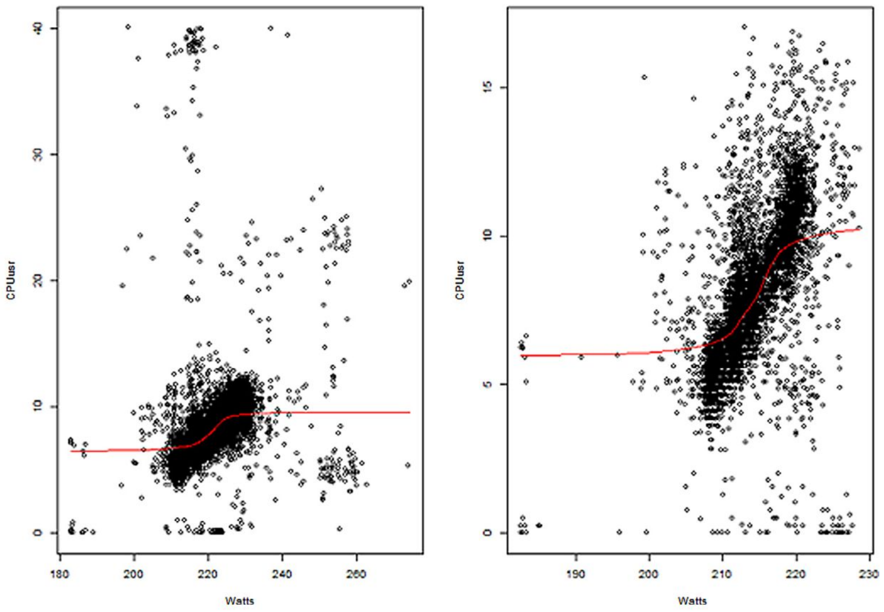


Figure: CPU utilization and CPU Energy Consumption before and after applying Practice 1

Outline

- Energy Efficiency Across **Programming Languages**
- Empirical Evaluation of **Two Best Practices** for Energy-Efficient Software Development

Measurement-Based

- Catalog of **Energy Patterns** for **Mobile** Applications

Data Mining

- An Approach Using Performance **Models** for Supporting Energy Analysis of Software Systems
- An independent assessment and improvement of the **Digital Environmental Footprint formulas**

Model-Based

Catalog of Energy Patterns for Mobile Applications

Home > Empirical Software Engineering > Article

Published: 05 March 2019

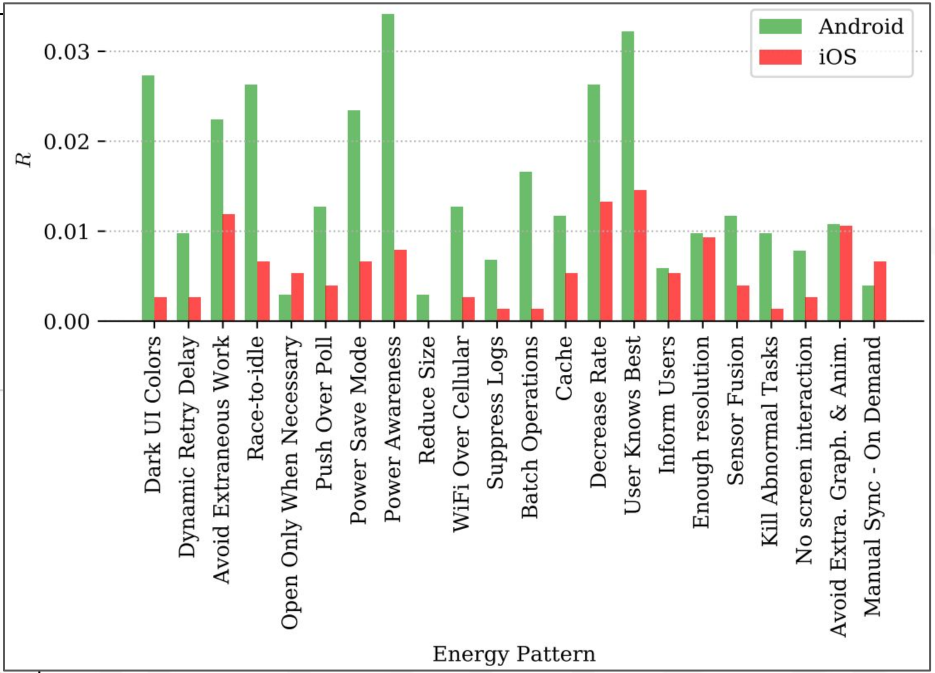
Catalog of energy patterns for mobile applications

Luis Cruz & Rui Abreu

Empirical Software Engineering **24**, 2209–2235 (2019) | Cite this article

Abstract

Software engineers make use of design patterns for reasons that range from performance to code comprehensibility. Several design patterns capturing the body of knowledge of best practices have been proposed in the past, namely creational, structural and behavioral patterns. However, with the advent of mobile devices, it becomes a necessity a catalog of design patterns for energy efficiency. In this work, we inspect commits, issues and pull requests of 1027 Android and 756 iOS apps to identify common practices when improving energy efficiency. This analysis yielded a catalog, available online, with 22 design patterns related to improving the energy efficiency of mobile apps. We argue that this catalog might be of relevance to other domains such as Cyber-Physical Systems and Internet of Things. As a side contribution, an analysis of the differences between Android and iOS devices shows that the Android community is more energy-aware.



Catalog of Energy Patterns for Mobile Applications

Motivation:
The adoption of ***design patterns*** is widespread across software developers, e.g., to **avoid performance bottlenecks and increase comprehensibility**

Method:

Mining software repositories: inspect commits, issues and pull requests on GitHub

Research Questions

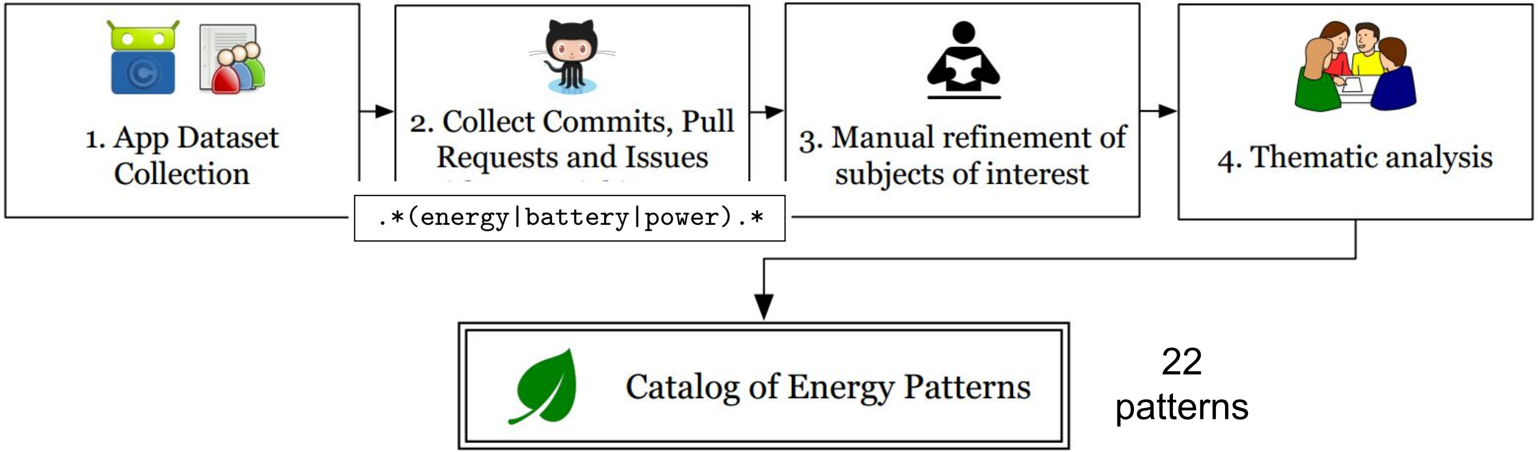
RQ1: Which design patterns do mobile app developers **adopt** to improve energy efficiency?

RQ2: How different are mobile app **practices** addressing energy efficiency **across** different **platforms**?

Catalog of Energy Patterns for Mobile Applications

Design Pattern: Each pattern describes a ***recurrent*** design problem, its **solution** and the **consequences** of applying it

1027 Android apps (F-Droid) and 756 iOS apps (Collaborative List of Open-Source iOS Apps)



22 patterns

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995), *Design patterns: elements of reusable object-oriented software*, Pearson Deutschland

Dataset Collection

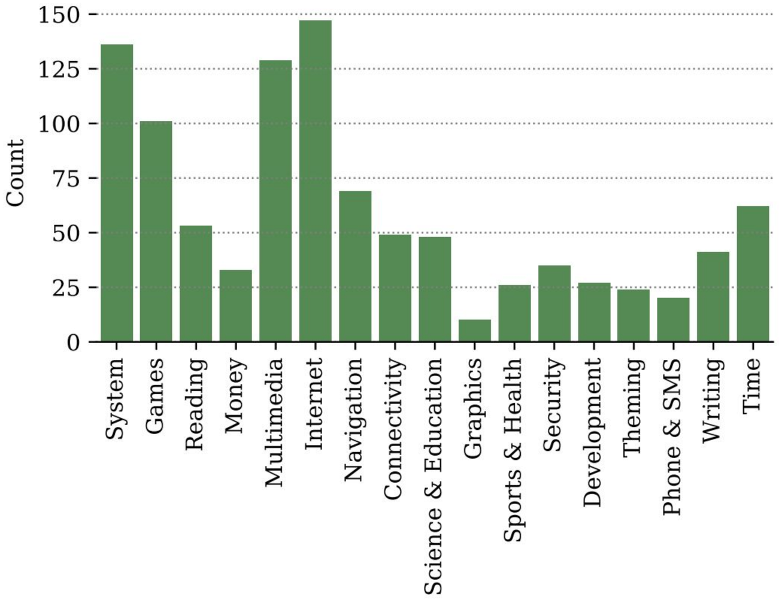


Figure: Android Applications Categories

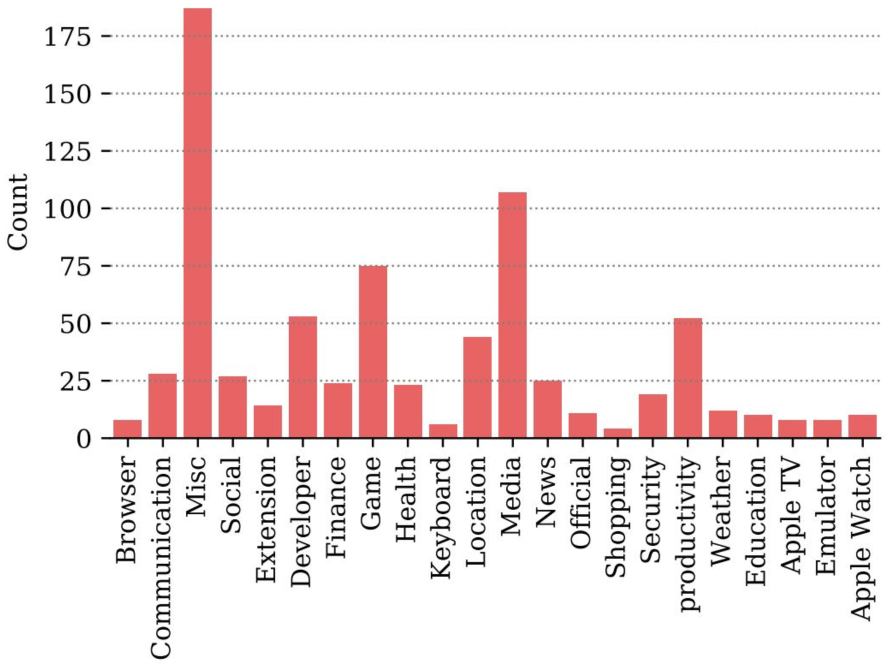


Figure: iOS Application Categories

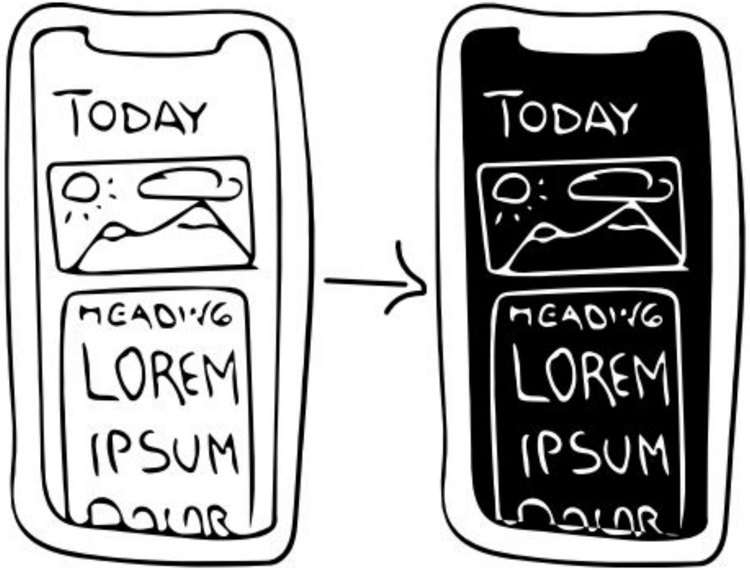
1. https://f-droid.org/
2. https://github.com/dkhamsing/open-source-ios-apps

Dark UI Colors

Context:
Apps that require heavy usage of screen (e.g., reading apps) can have a substantial negative impact on battery life

Solution:
Provide a UI with dark background colors

Example:
Provide a theme with a dark background using light colors to display text.

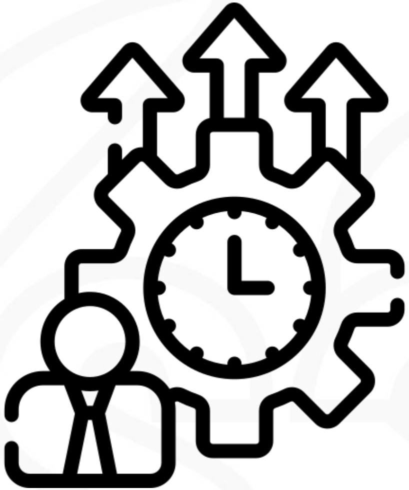


Dynamic Retry Delay

Context:
A resource is unavailable, the app will unnecessarily try to connect the resource for a number of times, leading to unnecessary power consumption.

Solution:
Increase retry interval after each failed connection

Example:
Instead of continuously polling the server until the server is available, use the Fibonacci series to increase the time between attempts

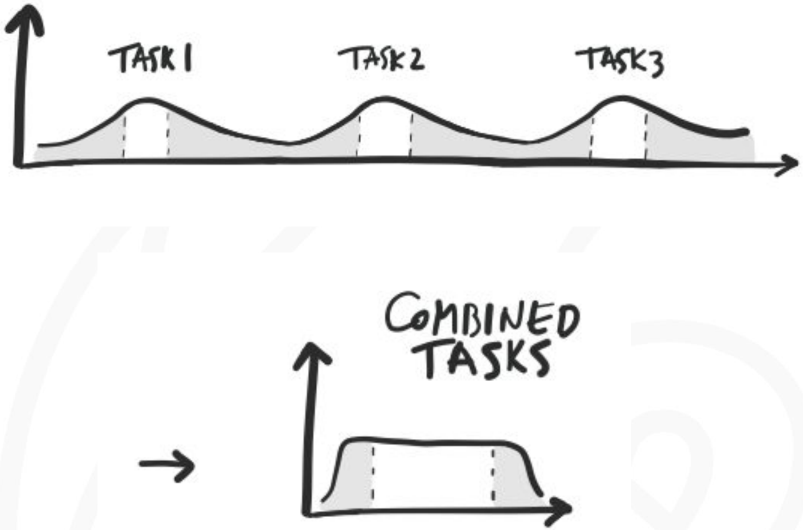


Batch Operations

Context:
Executing operations separately leads to extraneous tail energy consumptions

Solution:
Bundle multiple operations in a single one. By combining multiple tasks, tail energy consumptions can be optimized

Example:
Use Job Scheduling APIs (e.g., 'android.app.job.JobScheduler', 'Firebase JobDispatcher') that manage multiple background tasks occurring in a device.



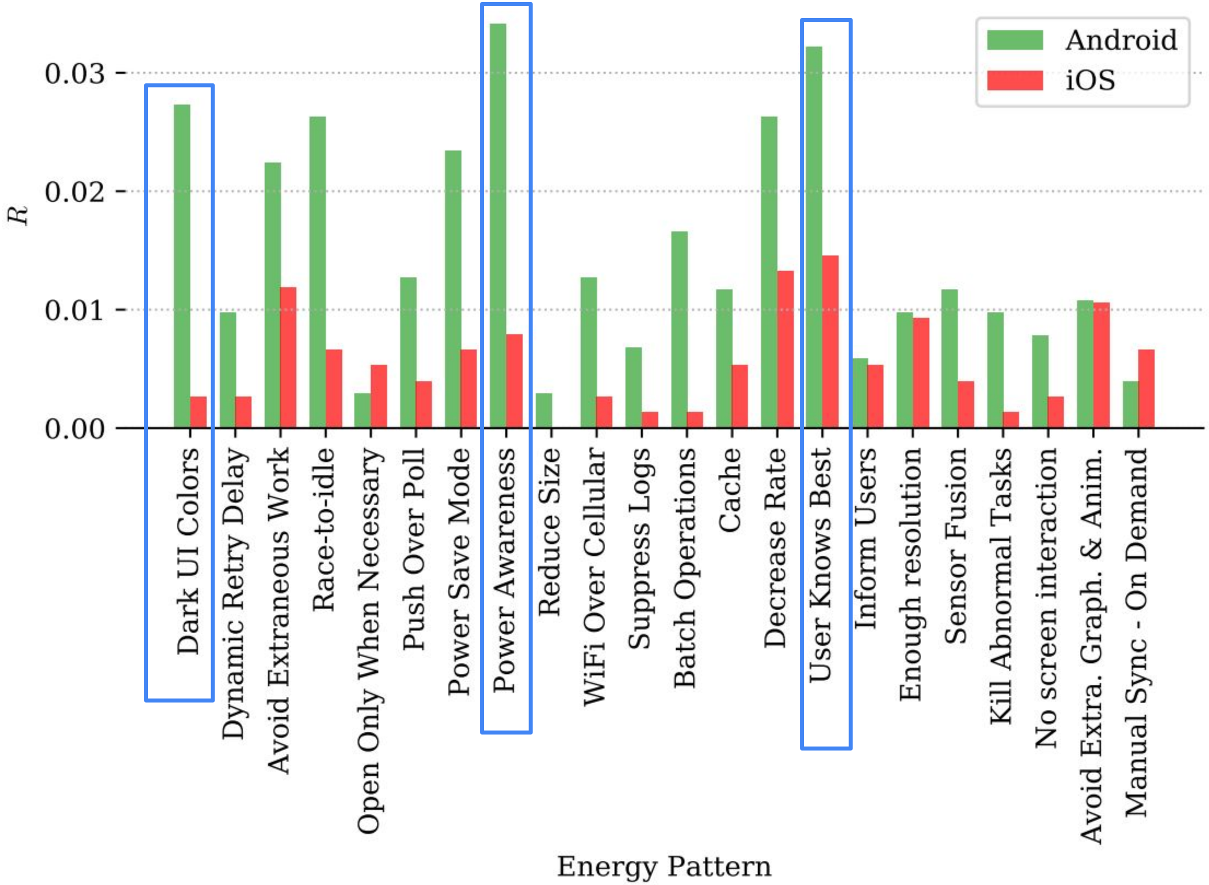
Cache

Context:
Same data is being collected from the server multiple times

Solution:
Implement caching mechanisms to temporarily store data from a server

Example:
Instead of downloading basic information and profile pictures every time a given profile is opened, the app can use data that was locally stored from earlier visits

Energy Patterns Frequency



Insights

- **Patterns** found in 133 Android apps (13%) and 28 iOS apps (4%)
    - Reasons not deeply discussed in the study (App Store constraints)
- **Characteristics** of the **applications** can have **influenced** the results
    - Sample unbalanced
    - Technology (e.g., AMOLED Screen)
    - APIs Features (e.g., Batch Operations in Android)
- There is no empirical study that has evaluated the cost and benefit of applying these patterns

Outline

- Energy Efficiency Across **Programming Languages**
- Empirical Evaluation of **Two Best Practices** for Energy-Efficient Software Development

Measurement-Based

- Catalog of **Energy Patterns** for **Mobile** Applications

Data Mining

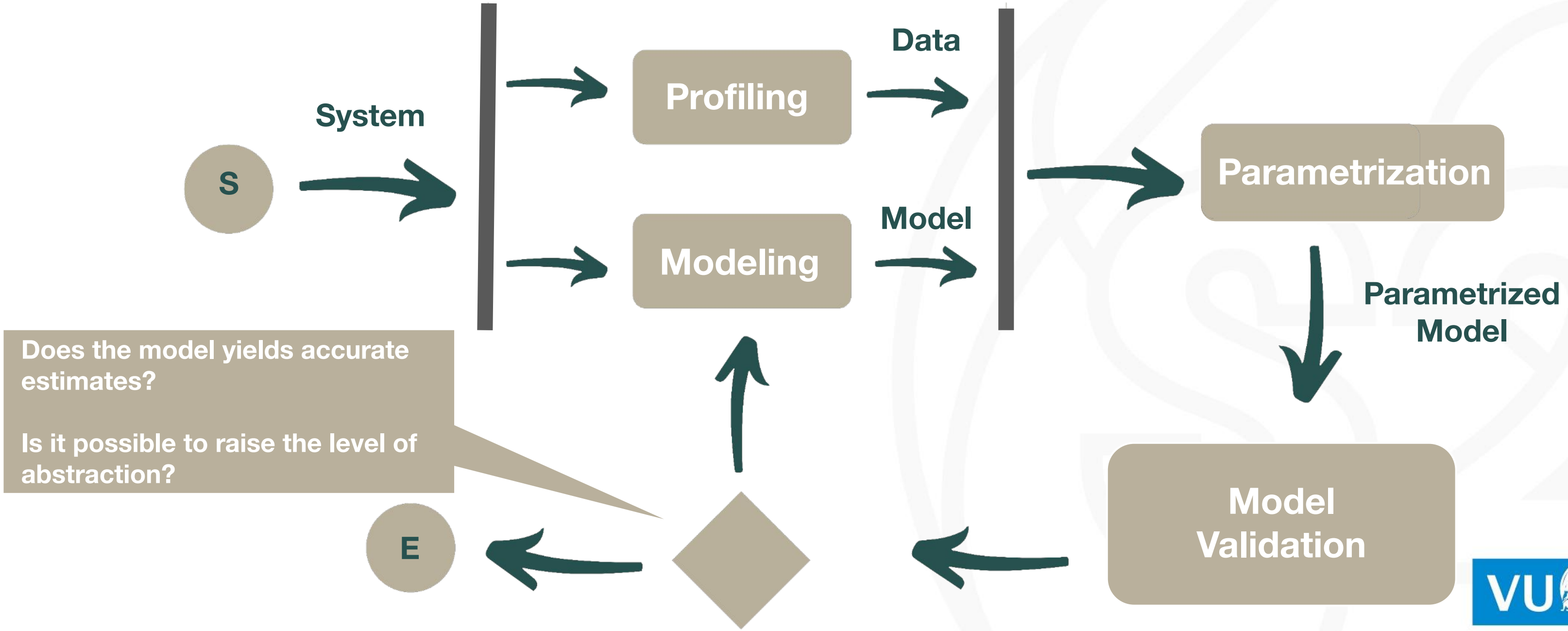
- An Approach Using Performance **Models** for Supporting Energy Analysis of Software Systems
- An independent assessment and improvement of the **Digital Environmental Footprint formulas**

Model-Based

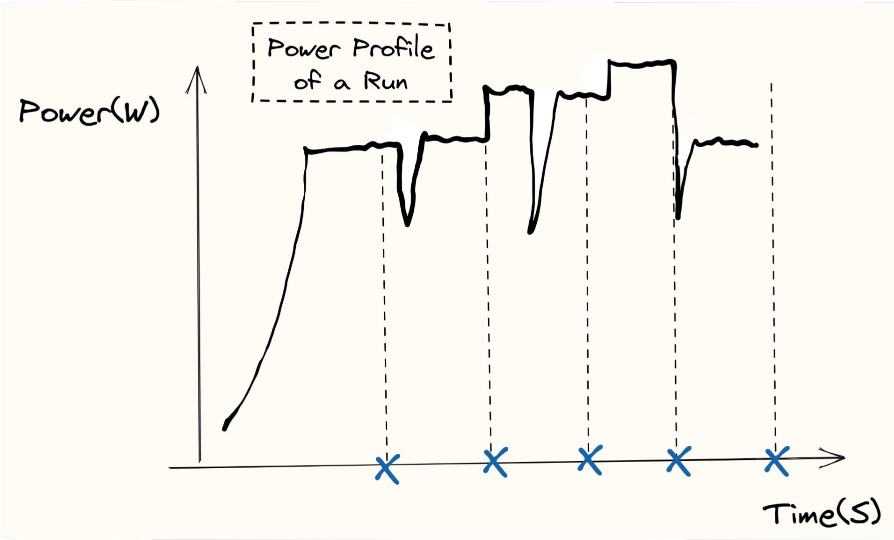
VU VRIJE UNIVERSITEIT AMSTERDAM

Reducing the Reality Gap

Explore the ***combination*** of measurement-based experiments and modeling in the context of ***energy/performance*** *analysis* of software systems

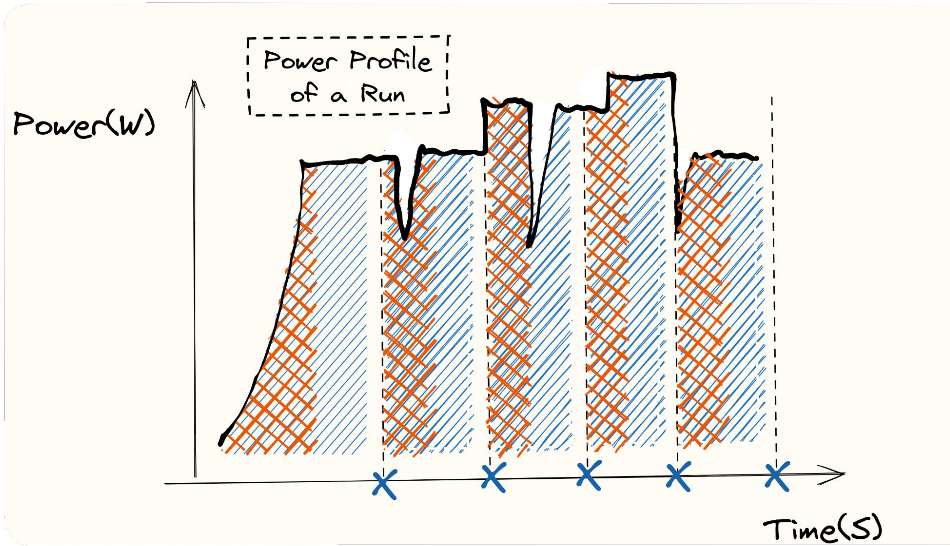


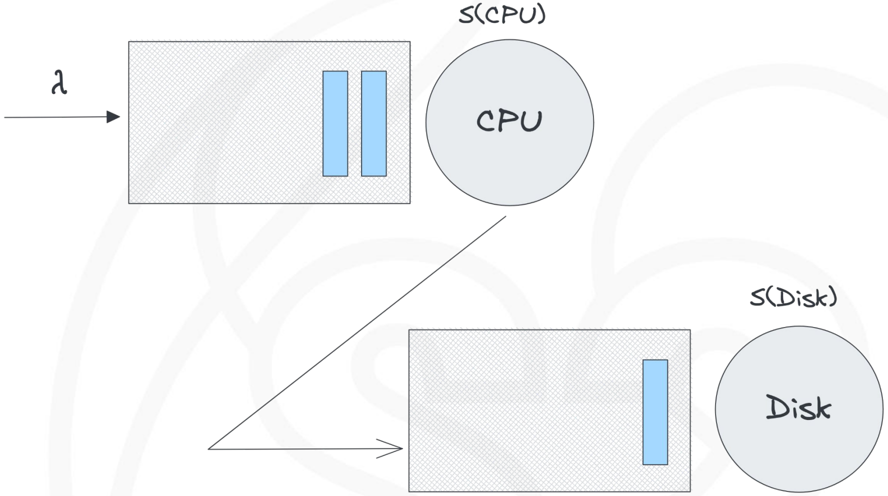
Power Profile



1. Behavior(Model) ~ Behavior(System)
2. Behavior → PowerProfile
3. PowerProfile(System) ~ PowerProfile(Model)

Queuing Networks





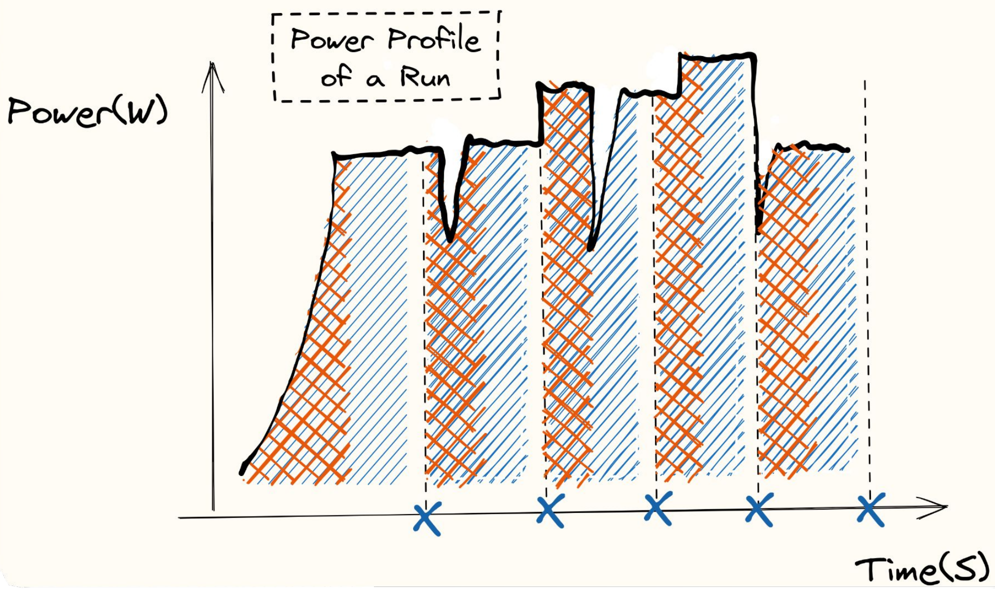
CPU-Time

Disk-Time

X

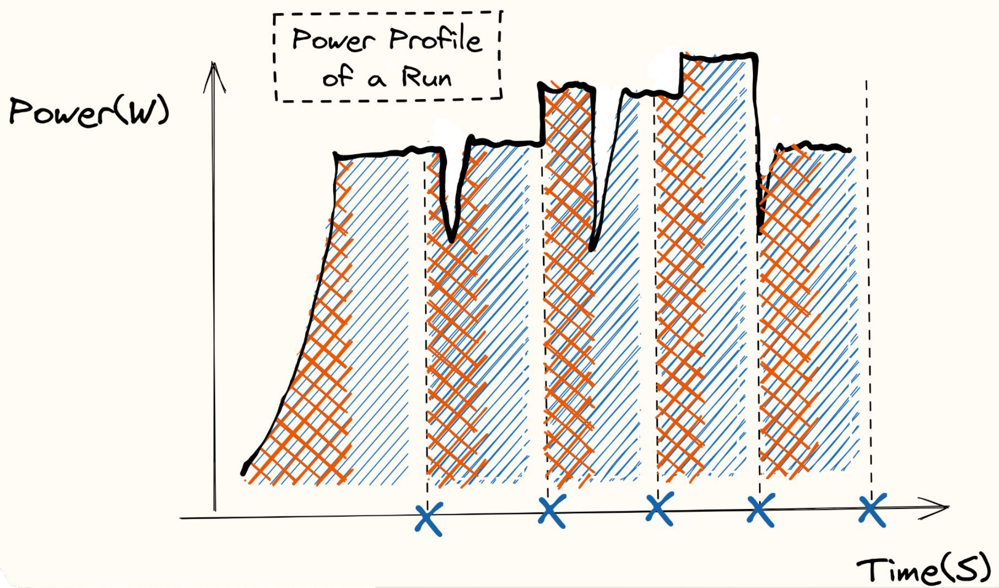
Observation Time

Resources Average Power Consumption



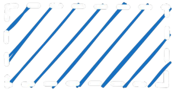
$$E(res, i) = \int_{t0,i}^{S_{res}} P(t)\, dt [\frac{Joule}{Visit}] \quad (1)$$

Resources Average Power Consumption

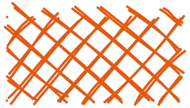


$$E(res, i) = \int_{t0,i}^{S_{res}} P(t)\, dt\left[\frac{Joule}{Visit}\right] \quad (1)$$

$$ED(res) = \sum_{i=1}^{\#Visit} \int_{t0,i}^{S_{res,i}} P(t)\, dt[Joule] \quad (2)$$



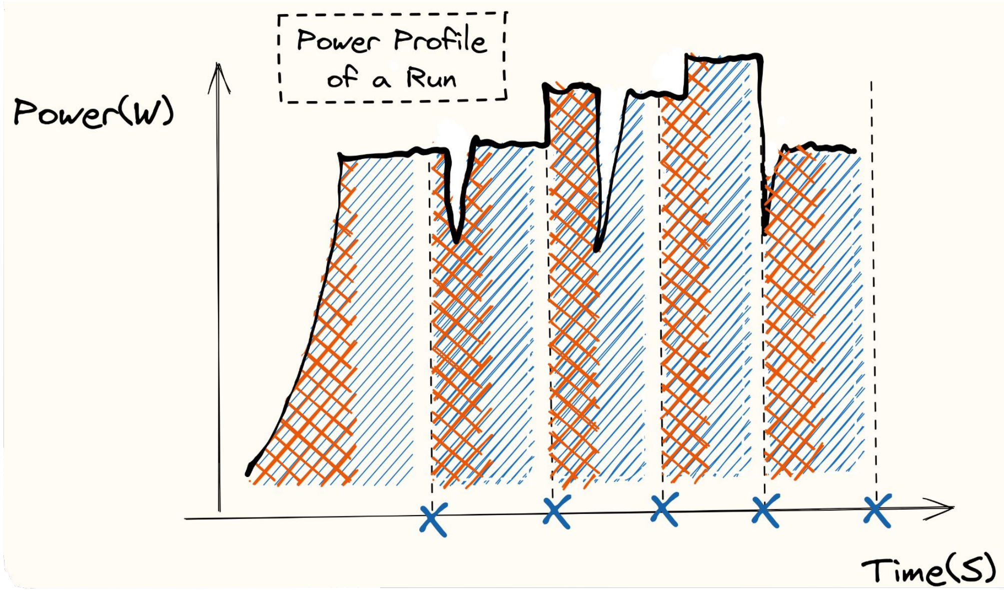
CPU-Time



Disk-Time

Observation Time

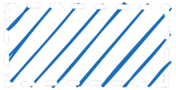
Resources Average Power Consumption



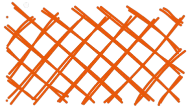
$$E(res, i) = \int_{t0,i}^{S_{res}} P(t)\, dt[\frac{Joule}{Visit}] \quad (1)$$

$$ED(res) = \sum_{i=1}^{\#Visit} \int_{t0,i}^{S_{res,i}} P(t)\, dt[Joule] \quad (2)$$

$$E(res) = \frac{ED(res)}{\#Visit}[\frac{Joule}{Visit}] \quad (3)$$



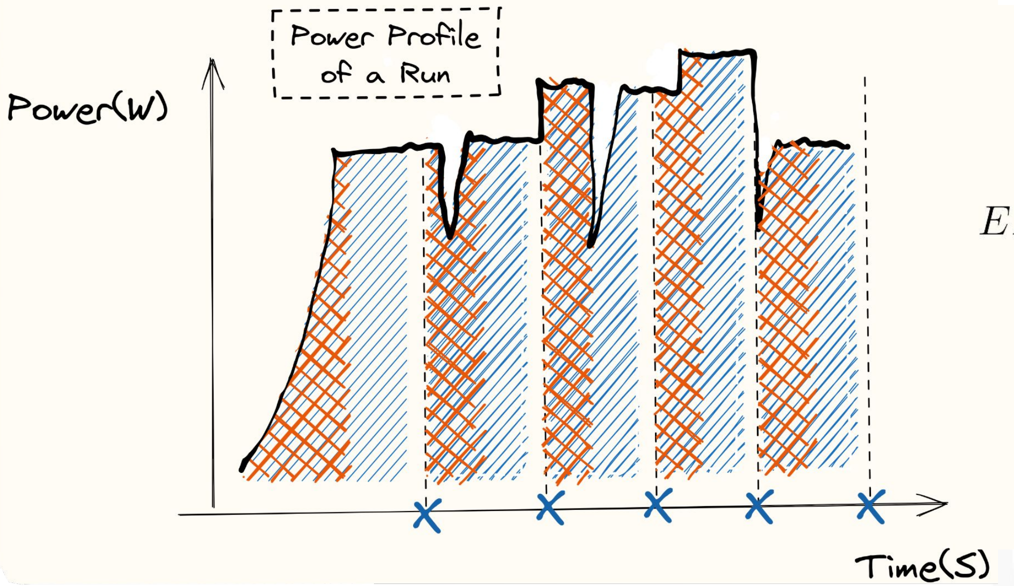
CPU-Time



Disk-Time

X Observation Time

Resources Average Power Consumption



$$E(res, i) = \int_{t0,i}^{S_{res}} P(t)\,dt\left[\frac{Joule}{Visit}\right] \quad (1)$$

$$ED(res) = \sum_{i=1}^{\#Visit} \int_{t0,i}^{S_{res,i}} P(t)\,dt[Joule] \quad (2)$$

$$E(res) = \frac{ED(res)}{\#Visit}\left[\frac{Joule}{Visit}\right] \quad (3)$$

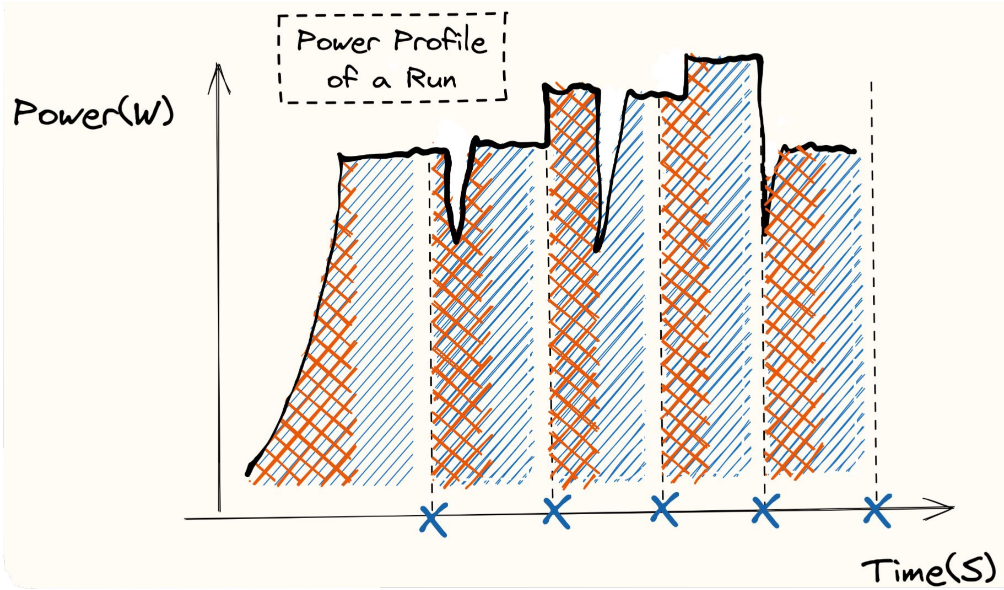
$$e(res) = \frac{E(res)}{S(res)}\left[\frac{Joule}{s}\right] \quad (4)$$

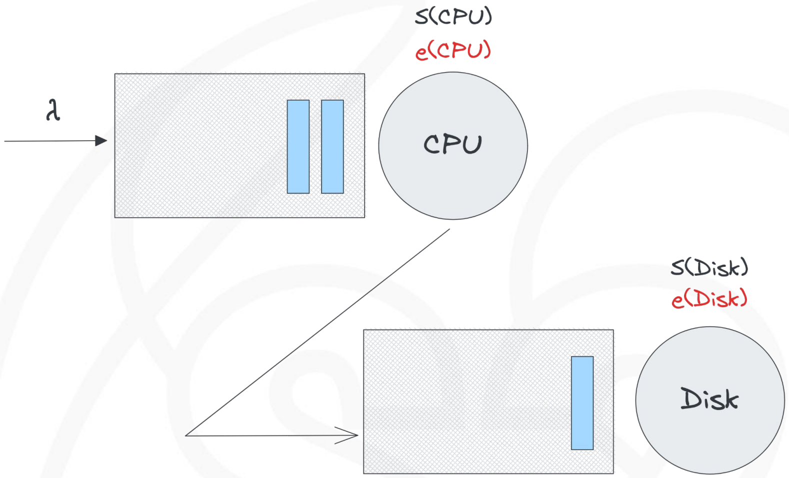
CPU-Time

Disk-Time

Observation Time

Resources Average Power Consumption





Resources Average Power Consumption

Two Case Studies:

Digital Camera [3]

Train Ticket Booking System [4]

For each case:

1. Observe the system under **scaled** workloads
2. Create a Layered Queuing Network (LQN) parametrized with measures obtained in the **shortest** experiment
3. **Compare** estimates vs measurements

Our approach, at the moment, considers only the cases in which *energy consumption **grows linearly** with execution time*

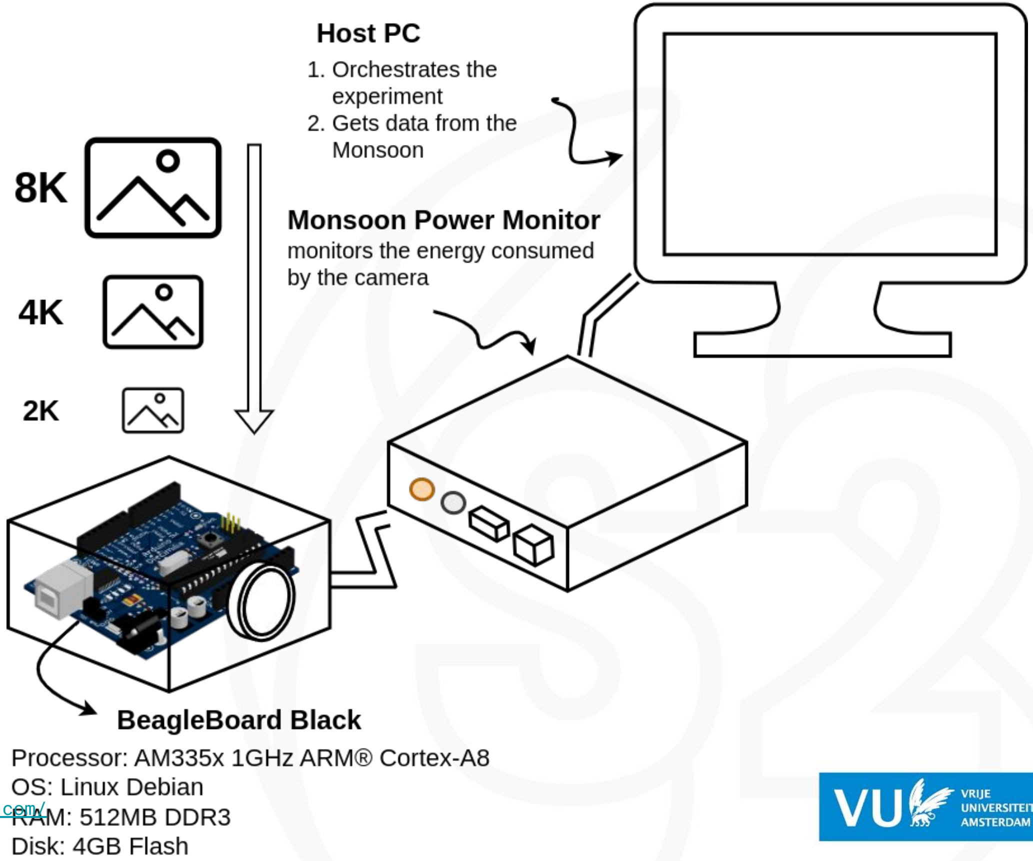
F. Vahid and T. Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*, John Wiley & Sons, Inc., USA, 1st edition, 2001
Fudan Software Engineering Laboratory, *Train Ticket Booking System*, https://github.com/FudanSELab/train-ticket, accessed: 2023-04-12

VU VRIJE UNIVERSITEIT AMSTERDAM

Digital Camera

A total of **thirty batches** are provided to the application, i.e., 10 per format.

A batch contains **30 pictures** of the same format chosen between 2K, 4K, and 8K



[5] Monsoon Solutions Inc, *Monsoon Power Monitor*, https://www.msoon.com/

Digital Camera

Format	Response Time (s)	CPU Utilization (%)	e (J/s)	Average Energy (J)
2K	60.30 - 60.30	96.30 - 96.48	**1.57**	95.27 - 95.16
4K	240.36 - **240.30**	96.76 - 96.12	1.59	382.46 - **379.24**
8K	960.73 - **960.60**	97.39 - 96.06	1.59	1537.96 - **1516.04**

Cells presenting two values have measured value, on the left, and estimate, on the right

$$e(res) = \frac{E(res)}{S(res)}[\frac{Joule}{s}] \Rightarrow E(res) = e(res) \times S(res)[Joule]$$

Train Ticket Booking System

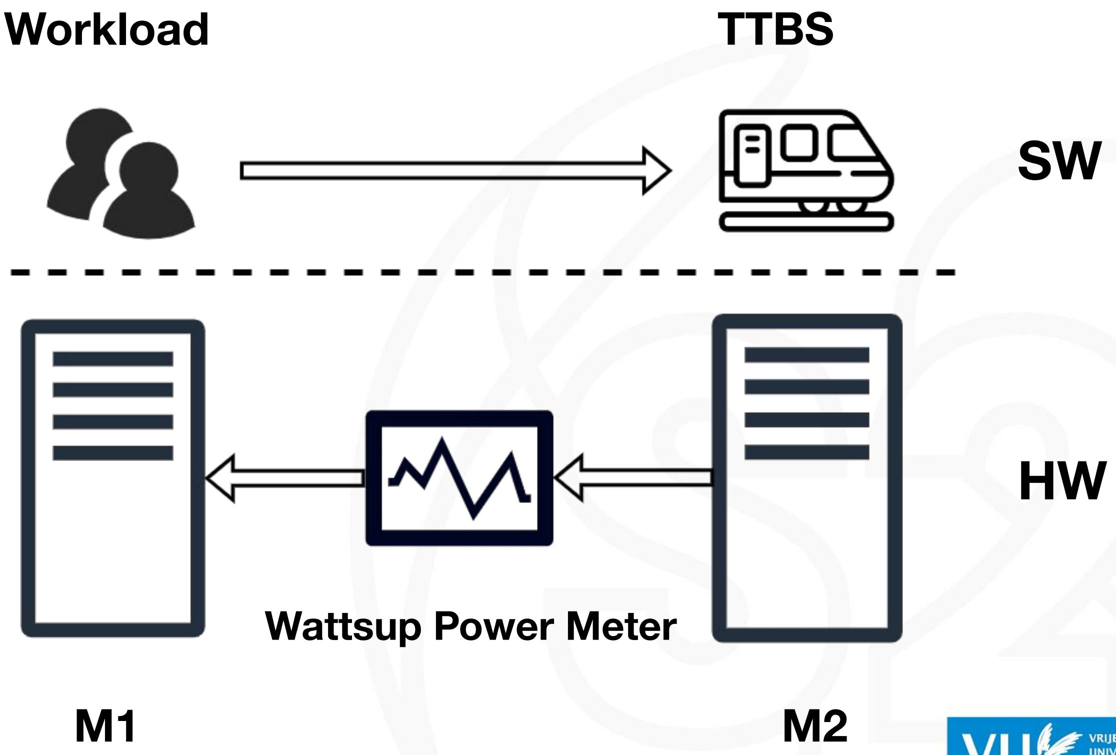
M2

Executes TTBS

M1

Generates **Bursts** of 75, 150, 225, 300, 375, 450, 500 Customers using **JMeter**

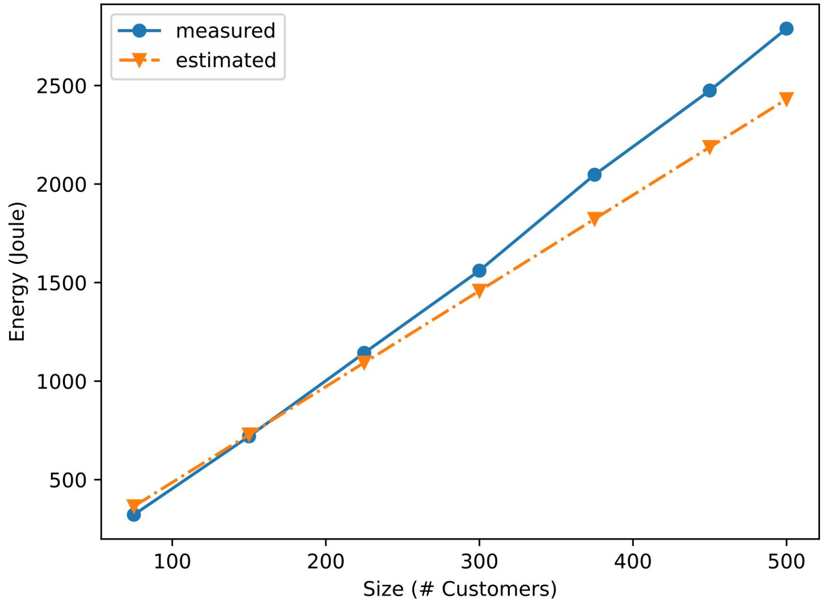
Records Performance and Power Consumption Values



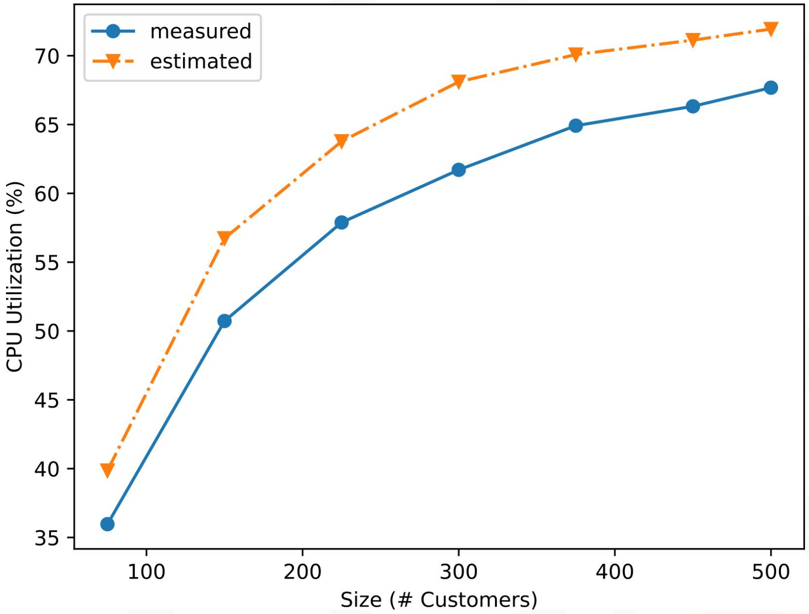
Power Meter Store, Watts Up Power Meter, https://www.powermeterstore.com/p1206/watts_up_pro.php

Train Ticket Booking System

Mean Absolute Percentage Error: (i) 9.24% **CPU Util.** (ii) 8.47% **Energy Consumption**
Experimentation Time: from 5 hours to 35 minutes



Energy Consumption



Performance

Limitations

Data sampled during the shortest experiment **can be not representative** of the SW

The set of data points evaluated in both case studies is **too small**

Future Work

Examine the performance and energy consumption of **resources other than the CPU**, such as the disk and network

Consider **different modeling notations** that could be more suitable in specific application domains

Consider cases in which CPU frequency and voltage are **dynamically adjusted** (DVFS)

VU VRIJE UNIVERSITEIT AMSTERDAM

Assessment and Improvement of DEF formulas

If direct measurements are impossible (Cloud), ***closed-form energy models*** can help quick **decision making** and **rough estimations**

Sustainable Digital Infrastructure Alliance (SDIA):

Digital Environmental Footprint ***(DEF)*** set of formulas for *energy consumption estimation* of **software services**

$$E_{tot} = E_{cpu} + E_{mem} + E_{IO} + E_{net} + \beta_{idle}$$

$$E_{tot} = U_{cpu} f_{cpu}(U_{cpu}) + U_{mem} f_{mem}(U_{mem}) + U_{IO} f_{IO}(U_{IO}) + U_{net} f_{net}(U_{net}) + \beta_{idle}$$

Tom Kennes, Measuring IT Carbon Footprint: What is the Current Status Actually? https://arxiv.org/abs/2306.10049

Energy Model

A1: We can use the Thermal Design Power (TDP) to indicate the energy consumption of a server when full load is applied to the CPU

A2: $\alpha_{CPU} + \alpha_{mem} + \alpha_{IO} + \alpha_{net} = 1$

A3: The energy consumed by a server increases linearly relative to the increase in usage of any of its components

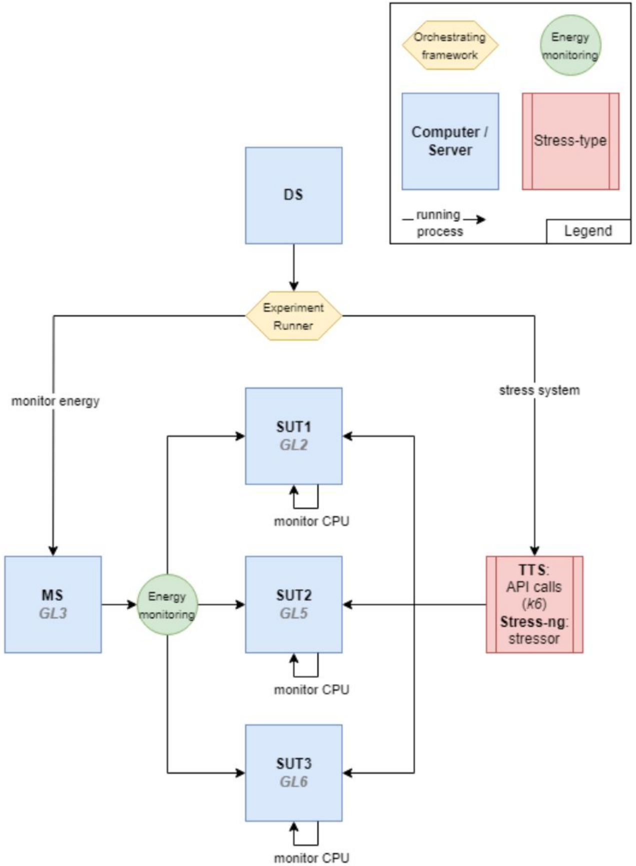
A4: Idling consumption of resources is expected to be zero

$$E_{CPU\ max} = U_{CPU\ max} * f_{CPU}(U_{CPU\ max}) = N_{CPU} * TDP$$

$$E_{tot\ max} = E_{CPU\ max}/\alpha_{CPU} = [N_{CPU} * TDP]/\alpha_{CPU}$$

$$E_{tot\ predict} = CPU_{workload}\% * [N_{CPU} * TDP]/\alpha_{CPU}$$

Experiment Setup



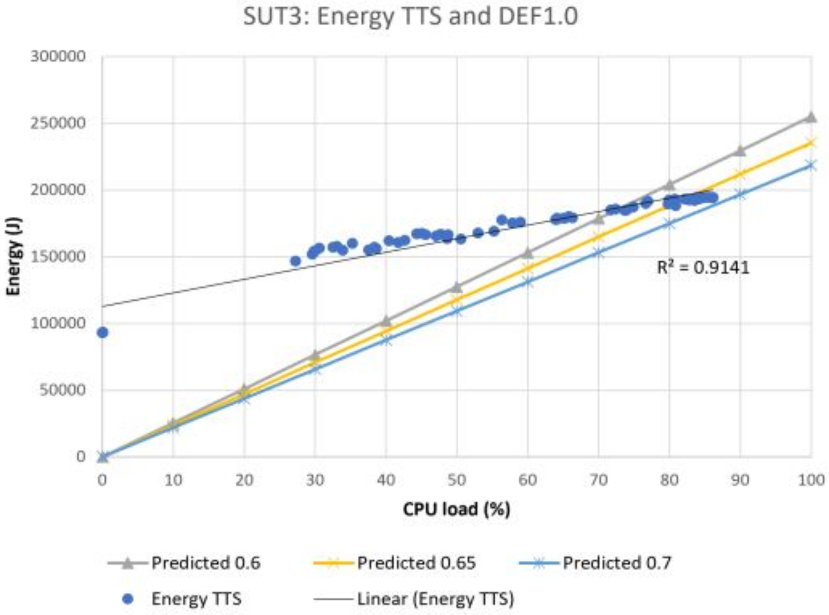
- Verify the **accuracy** of the DEF formulas
- The results were validated using two different workloads (1 synthetic and 1 realistic)
  - *Synthetic* = stress-ng
  - *Realistic* = Train-Ticket Booking System + k6
- ***Independent Variable:*** %CPU
  - Treatments: Idle, 50%, 75%, 100%
- ***Dependent Variable:*** Energy Consumption
- 10 *Run* per Treatment of 15 *Minutes*
- 5 minutes **cooling time** *between measurements*

Results

DEF 1.0 $E_{tot\ predict} = CPU_{workload}\% * [N_{CPU} * TDP]/\alpha_{CPU}$

a_{CPU} fixed to {0.6, 0.65. 0.7}

- Linearity between energy consumption and CPU Load
- Energy Consumption Average Error Rate (%): *14.04 - 17.74%*
- MAX Avg Error Rate (%): *13.96% - 32%*



Refinement

DEF 1.0

$$E_{tot\ predict} = CPU_{workload}\% * [N_{CPU} * TDP]/\alpha_{CPU}$$

DEF 2.0

$$Pidle_{CPU} = Ridle * Pcpu = 0.28 * N_{CPU} * TDP$$

DEF 2.1

$$Pidle_{MAX} = (0.28 * TDP * Ncpu)/\alpha_{CPU}$$

$$Ptot = (Pmax - Pidle_{CPU}) * \%CPU + Pidle_{CPU}$$

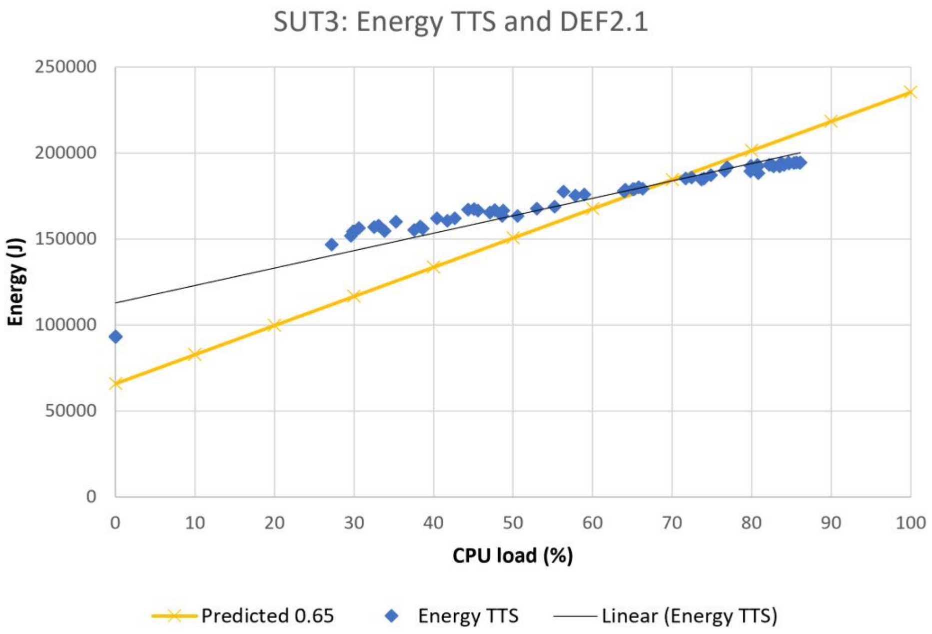
$$E_{tot} = P_{tot} * t$$

Results

DEF 2.X $Ptot = (Pmax - Pidle_{CPU}) * \%CPU + Pidle_{CPU}$

$$E_{tot} = P_{tot} * t$$

- Linearity between energy consumption and CPU Load
- Energy Consumption Average Error Rate (%):
  - *DEF2.0: 12.36 - 13.96%*
  - *DEF2.1: 11.08 - 15.42%*
- Best Results with a_{CPU} fixed to {0.6, 0.65}



Thanks!
Any Questions?

email: v.stoico@vu.nl

IMG: Keith Haring, Untitled (Earth Day), 1985, https://emergencyartmuseum.com/haring