A Green Miner's Dataset: Mining the Impact of Software Change on Energy Consumption

Chenlei Zhang
Department of Computing Science
University of Alberta
Edmonton, Canada
chenlei.zhang@ualberta.ca

Abram Hindle
Department of Computing Science
University of Alberta
Edmonton, Canada
abram.hindle@ualberta.ca

ABSTRACT

With the advent of mobile computing, the responsibility of software developers to update and ship energy efficient applications has never been more pronounced. Green mining attempts to address this responsibility by examining the impact of software change on energy consumption. One problem with green mining is that power performance data is not readily available, unlike many other forms of MSR research. Green miners have to create tests and run them across numerous versions of a software project because power performance data was either missing or never existed for that particular project. In this paper we describe multiple open green mining datasets used in prior green mining work. The dataset includes numerous power traces and parallel system call and CPU/IO/Memory traces of multiple versions of multiple products. These datasets enable those more interested in data-mining and modeling to work on green mining problems as well.

Categories and Subject Descriptors

D.4.8 [**Performance**]: Energy; D.2.5 [**Testing**]: Regression

General Terms

Keywords

Software Energy Consumption; Software Change; Dataset

1. INTRODUCTION

As software changes so does its performance profile. As features are added and bugs are fixed the performance of a software system tends to change. CPU use is easy to measure, but energy consumption performance is far more difficult to measure, often requiring hardware support or instrumentation. This is a barrier for developers who are concerned about power performance regressions.

A concrete body of research has been applied to build power models for applications on mobile devices. Zhang et al. [10] have implemented a power model for Android smartphones, PowerTutor, which is based on the power modeling of each hardware component. Dong et al. [1] have applied a different approach to modeling application power consumption for Linux-based mobile systems based on the system statistics of each hardware component. Gupta et al. [3] have studied the power consumption of Windows phone. They combined power traces and execution logs in Windows phone to build power models. Pathak et al. [7] have generated power consumption finite state machines for components on smartphones and developed power profiler to estimate power consumption of applications. The most recent study was done by Hao et al. [4] regarding their power model, *eLens*, which models Android applications based on Java instructions. These studies could help developers account for their applications energy consumption. However, the impact of software change on software energy consumption was not addressed.

Green mining [5, 6] is the study of how software energy consumption (sometimes referred to as software power consumption) and software power use relate to software maintenance. Green mining asks, "how does software change impact the energy consumption profiles of a software product?"

In this data paper, we focus on generating the datasets that correlate software change and energy consumption. Software energy consumption could be measured by a power meter. In terms of software changes, we made use of system calls, which is an essential interface sitting between the application and the kernel of operating system (OS) that triggers hardware utilization and other kernel services [7]. We can expect different versions of software to invoke different system calls during their execution if they differ from each other in which services to get and how to get the services from the OS kernel. We also measure the use of CPU, IO and Memory.

To be specific, our datasets consist of software energy consumption as well as the number of system call invocations or resource use for 5 applications under various test cases across versions. Ultimately we hope this dataset can help MSR researcher and developers understand their impact on software energy behaviour while maintaining software.

2. DATASETS

In this section, we explain the organization of our datasets and its potential usage. Our tracing datasets were generated by analyzing two open source applications, the text editor `gedit`, and the audio player `mpg123`. For each application, we have built multiple versions and developed two test cases to gather the data. The data gathered from each test case forms a dataset and it contains the mean power use and the corresponding invocation count of system calls for each version. Each dataset is in CSV format and each row in the CSV file represents the data of each application version. The number of columns in each dataset varies because of the number of different system calls traced in different applications as well as test cases. In our trace datasets, we have also

MSR '14, May 31 – June 1, 2014, Hyderabad, India

Table 1: Description of the datasets.

App	Test Case	#Versions	#SysCalls
gedit	text editing	39	79
	syntax highlighting	39	77
mpg123	MP3 playing	68	56
	stream playing	47	60
Firefox	reading webpages	509	N/A
electrolysis	reading webpages	582	N/A
Vuze	leeching bittorrent	45	N/A
	idling bittorrent	45	N/A
rTorrent	leeching bittorrent	40	N/A

Table 2: Schema for Trace-based datasets `gedit` and `mpg123`

Field	Description
POWER	mean wattage of the test run
getsockname	count of getsockname calls
send	count of send calls
...	
x	count of x calls where x is a syscall

included two text files that store the version number we tested on for both applications. There is some variation regarding the number of different system calls traced under two test cases for each application. But the first column is always the mean power consumption of each application version ordered chronologically. The rest of the columns are different system calls and each entry shows the number of system call invocations. System calls are described within the Linux man page [8]. An overview of the schema is available in Table 2. This trace-based dataset is publicly available at https://github.com/greentrace/green-dataset.

For datasets without system calls that were used in our prior work [6], we created 4 kinds of tests for 3 products: the web browser Firefox, and the BitTorrent clients Vuze and rTorrent. Repeatable tests were run against these systems numerous times, while their power use and resource usage statistics were recorded. Uncompressed it is approximately 1GB in size due to large number of per second measurements taken for each and every test. These readings and aggregates are stored in available in CSV files available at https://github.com/abramhindle/green-data-msr/. The per second measurements also include system level CPU, IO, and Memory statistics extracted with SAR [2], joined together. An overview of the per second schema is available in Table 3 while the aggregated schema is described in Table 4. This means that one can correlate resource use with power use.

Table 1 summarizes our datasets in terms of the number of application versions and the number of traced system calls. For gedit there are 39 versions tested with both text editing and syntax highlighting test cases (we describe how to develop test cases in Section 3). For mpg123, we have tested on 68 versions under MP3 playing test case and 47 versions under stream playing test case.

2.1 Potential Use

With our datasets, one can study the power behaviour of an application across multiple versions. By comparing the mean power consumption of an application over versions, one can visualize the changing trend of the application energy consumption. More importantly, system calls act as the entry points into the OS kernel for user applications. Thus system call invocations have the potential of modeling software energy consumption based on multiple versions and also trace back to software in order to locate software changes that are responsible for energy consumption variations using system calls. For the green mining datasets without traces, SAR has provided fine grained resource utilization measures that can be used to develop power estimates or resource use estimates. We suggest our dataset should be combined with other datasets or static/dynamic analysis in order to mine for relationships between power use and design patterns, bug reports, fixed code, fix inducing code, churn, metrics, etc. Therefore, our datasets can be mined for discovering the specific relationship between OS use, resource use, software change, and software energy consumption.

Table 3: Partial Schema for per second traces of `Firefox`, `Vuze` and `rTorrent` (*-model.csv files)

Field	Description
FILE.testID	name of the exact test
FILE.sURI	binary tested
FILE.n	reading number for the test
now	unix time of the written record
Power.amps	amperage measured
Power.volts	voltage measured
Power.watts	watts measured
SAR.%system	% CPU time spent in kernel
SAR.%idle	% CPU time spent idle
SAR.%user	% CPU time spent in userspace
SAR.fault/s	Page Faults / Second
tps	transfers to disk / Second

Table 4: Partial Schema for per version aggregate measurements of `Firefox`, `Vuze` and `rTorrent` (*-aggregate.csv files)

Field	Description
machine	test computer name
utime	UNIX time of the written record
sURI	binary tested
ffversion	version of binary
kwh	energy consumed in KwH
max	maximum watts measured
min	minimum watts measured
mean	mean watts measured
var	variance of watts
seconds	test length in seconds

3. METHODOLOGY

In this section, we explain the methodology for collecting and parsing the software energy consumption among multiple versions of software. The general process is derived from the previous work on *Green Mining* [5].

1. Choose and build multiple versions of a software product.
2. Decide on the level of instrumentation.
3. Develop the test cases to run on the software.
4. Configure the testbed.
5. Run the tests and collect data.

3.1 Choosing and Building Multiple Versions of a Software Product

We have studied five software products in this paper. The first software project tested was gedit. This general purpose text editor has been developing for more than 10 years. It is written in C and Python, and is the default text editor for GNOME desktop. The second software project tested was mpg123. It is a command-line

MPEG audio player written in C and assembly. It has been actively maintained in the past 7 years.

In order to build multiple versions of `gedit`, we relied on the Git repository of `gedit` [1]. When choosing commits to compile `gedit`, we only considered releases. From release 0.7.9 to release 3.7.3 (commits starting from June 2000 to February 2013), we were able to build 39 revisions of `gedit`, covering most of the `gedit` 2 (18 builds) and `gedit` 3 versions (21 builds).

For `mpg123` we built the binaries from the tarballs of past versions. In total, we have 68 reversions of `mpg123` covering most of the versions from 0.66 to 1.15.4.

For `Firefox` we relied on 509 nightly snapshots for the 2009–2010 nightly builds of Firefox with versions ranging from 2.0 to 3.6 and 482 nightly snapshots from the `electrolysis` branch. For `Vuze`, we built 45 subversion revisions starting from revision 26730 on September 14, 2011 to revision 26801 on December 15, 2011 (3 months). For `rTorrent` we built 18 snapshot versions: `rTorrent` version 0.3.0 (2005) to `rTorrent` 0.8.9 (2011), and `libTorrent` versions 0.6.4 (2005) to 0.13.0 (2011). These three products were discussed in our prior work [6].

3.2 Deciding on the Level of Instrumentation

We recorded the power use and system calls consumed and invoked by the tested software products for each revision. The device we used to measure the energy consumption of the testbed is an AC power monitor, *Watts Up? Pro* [5]. This meter can continuously monitor and collect power measurement with an accuracy of $\pm 3\%$. This hardware can monitor real-time electricity usage and collect a variety of data, including power use in watts, and transmit this result over a USB-serial connection.

For recording system calls on the testbed, we applied the debugging utility for Linux, *strace* [2]. It is able to trace the name of each system call, its arguments and its return value called by a process or a program. Figure 1 shows a part of *strace* output for the Linux command `date`. The listed system calls invoked by a program could help us detect unexpected behaviours and thus *strace* is often used to debug programs.

For the datasets used in our prior work [5] (`Firefox`, `Vuze`, and `rTorrent`), we did not record system calls but we relied on SAR [2], a system monitoring utility, to record and monitor CPU, IO and Memory information.

```
read(3, "\nMST7MDT,M3.2.0,M11.1.0\n", 4096) = 24
close(3)                                = 0
munmap(0xb7715000, 4096)                = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 6), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7715000
write(1, "Tue Jul  2 19:35:57 MDT 2013\n", 29) = 29
close(1)                                = 0
munmap(0xb7715000, 4096)                = 0
close(2)                                = 0
```

Figure 1: An example of *strace* partial output for the Linux command `date`.

3.3 Developing the Test Cases

Changes in software are often scattered across various features and files. In order to correlate software changes with software energy consumption, we need to implement a set of test cases that are able to trigger different features. In this study, we developed two test cases for both `gedit` and `mpg123`.

For `gedit`, the first test case is about text editing. The second one is about syntax highlighting. `gedit` is able to highlight syntax for a number of program languages and text markup formats.

The test scenario for text editing is to simulate a user creating a new document and then typing text into it and finally saving the document. We built a X11::GUITest UI driver to simulate the mouse actions and typing actions that we pre-recorded based on our first author typing in the preamble of the GNU General Public License (GPL) [3]. The test took almost 6 minutes to type about 560 words of the preamble in the GNU GPL. The test procedure is to 1) start the application, which opens a new document; 2) type the GNU GPL Preamble; 3) save the file; and, 4) close the application.

The test case of syntax highlighting intends to simulate a real user reading through a variety of programming and text markup language code. There are six files to read and each file is source code (C, Java, Perl, and Python) or text from a markup language document (HTML and LaTex) that has more than 300 lines. The test took more than 7 minutes to go through all the six files. The test procedure is 1) open the six files in `gedit`; 2) scroll down to go through the first file in every few seconds until reach the last line; 3) move to the next file and repeat step 2; and 4) close `gedit` when finished going through the last file.

For `mpg123`, the test cases are to play MP3 files and music stream using `mpg123`. The core functionality of `mpg123` is to decode MPEG audio files. So our first test case for `mpg123` is to listen to music by playing MP3 files using `mpg123`. `mpg123` is a command-line based player so we tested it within a GNOME Terminal. It was started with a 3-minute long song as a command-line parameter and would play the song as soon as it started and then it terminated once the song finished playing. In the second test case for `mpg123`, we intended to test another functionality: playing a music stream. We also started the test from a GNOME Terminal and passed the url of the music stream as a parameter to start `mpg123`. After playing the music stream for about 3 minutes we terminate `mpg123` by killing the `mpg123` process. Newer versions in our `mpg123` builds cannot play music stream stably so only 47 versions of `mpg123` were tested in this test case.

Our `Firefox` and `electrolysis` tests, from prior work [6], were X11::GUITest driven tests where `Firefox` would startup and view multiple webpages scrolling through them. Within 6 minutes 4 different webpages were visited: 2 Wikipedia pages, a mirror of the main-page and a page about the "Battle of Vukovar", and 2 NYAN-Cat pages (`http://nyan.cat/`) mirrored in different ways but hosted remotely by us. Each test cleaned up before and after itself and ensured it had not cached anything.

For `Vuze` we made an idle test and a file download or leech test The idle test measured Vuze's start-up, idling and file integrity check. The leech test downloaded a 2GB file from a seeder. Both were terminated after a set period of time, making each test take the same amount of time. The testbed was cleaned before and after each test.

For `rTorrent`, tested in our prior work [6] we repeated the Vuze leech test. The testbed was cleaned before and after each test.

3.4 Configuring the Testbed

We implemented our tests on a laptop computer: Lenovo ThinkPad X31. It runs 32-bit Ubuntu 12.04. Tests run for `rTorrent`, `Firefox`, and `Vuze` were run on the same machine running Ubuntu 11.04, using the same configuration. For `Vuze` and `rTorrent` we ran a seeder on another computer on the same local area network over Ethernet. To minimize the noise when measuring the energy consumption of tests, we turned off any services and automatic updates performed by the operating system. We also disabled the screen saver and left the screen on during the tests. Headphones were plugged in for `mpg123` test cases.

[1]Git Repo of `gedit`, `https://git.gnome.org/browse/gedit/`

[2]strace, `http://sourceforge.net/projects/strace/`

[3]GNU GPL, `http://www.gnu.org/copyleft/gpl.html`

We created a test-user, called greenmining, to run our test cases and this user would run the default Ubuntu Desktop. We removed the battery of the testing machine and it was plugged into a *Watts Up? Pro* power meter, which was instructed to continuously log its power consumption as RMS, with a resolution of 1 measurement-per-second. The data is recorded by GreenLogger, which is our application that records *Watts Up? Pro* readings.

3.5 Running the Tests and Collecting Data

For each test case in `gedit` and `mpg123`, we ran more than 10 tests over each software version we built. The first test was to trace the system calls and the rest of the tests were to measure the energy consumption of each software version. System calls for each version tend to be stable and we just traced them once. We chose 10 or more tests in order to determine normality and differences between power measurements.

Hence, for each software revision under each test case, there is one record of system calls and more than 10 records of energy consumption measurement. We took the mean of the multiple power use measurements for each software version under each test case. We also grouped the system calls by names and counted the number of invocations of each system call for each software version to form our datasets. These trace tests took more than 20 days to run.

For `Firefox` we had 43 distinct versions from 509 binaries, each binary had at least 3 test runs and each distinct version had at least 20 runs with a total of 2131 tests. For `electrolysis` we had 11 distinct releases over 482 binaries with a total of 1500 tests. For `Vuze` idle we ran 17 to 21 tests per each version of `Vuze`, for a total of 900 tests. For `Vuze` leech test we ran 10 to 15 tests per each of the 45 versions, resulting in 500 tests. For `rTorrent` we ran each test 6 to 10 times resulting in 294 tests across 40 combinations of `rTorrent` and `libTorrent`. These non-trace tests took more than 30 days to run.

4. TOOLS

We provide our tools used in this study to encourage other researchers to validate our datasets and also generate new datasets for other test cases as well as more applications. The tools are available at `https://github.com/greentrace/green-tools`.

For collecting data, we use **GreenLogger** with SAR [2] and **strace**, the open source debugging utility for Linux as we mentioned in the previous section. GreenLogger is able to collect power use readings from power meter *Watts Up? Pro*: one measurement per second. We also apply *strace* to get the summary of invocation counts for all the triggered system calls.

We have implemented the application **GreenTrace** for merging the collected two data sources. It converts the summary of invocation counts (tables separated by spaces) for all the system calls to a CSV file then merges it with the mean power consumption of multiple software versions.

5. LIMITATIONS

Since we utilized a power meter to measure the energy consumption of software and a UI driver to automate some of the test cases, the accuracy of power measurement in our datasets is limited by the accuracy of our power meter and the overhead from the UI driver. Tests are run multiple times because there are errors. Programs like Firefox are very large programs and they can often exhibit errors and bugs that are transient and inconsistent. Thus when evaluating runs of datasets be aware that some versions will have bugs that will make their power use seem extremely high or extremely low, these should be removed from analysis since there could have been numerous problems with the testbed, the software, the measurements etc. Most test runs were not observed by a human so many things could have happened.

The accuracy of traced system calls in our datasets is restricted by the assumption that system call invocations tend to be stable in each run of test case for the same software version.

Code coverage was not investigated when producing cases, not all changes and not all changed files are exercised by the tests. Changes that affect performance could be missed by our test cases.

The license of the data is CC-BY 4.0, attribution can be achieved by citing this paper or our other work [9, 6]. Academics are expected to cite the paper in works that use the data.

Issues that we did not control for include: internet connectivity and latency, temperature, light levels, and network traffic.

6. SUMMARY

Energy efficient applications are becoming more and more important for mobile computing platforms. Correlating the energy behaviour of applications to the historical data in software development repositories is a promising direction for revealing the impact of software change on energy consumption.

In this data paper, we provided the datasets for mining the impact of software changes on software energy consumption. We introduced the organization of our datasets gathered from multiple versions of `gedit`, `mpg123`, `Firefox`, `Vuze` and `rTorrent`. With well documented methodologies and tools we hope that our datasets proves useful to researchers interested in software power modeling without the hassle of actual measurement.

Despite of the limitations within our datasets, there are numerous potential usages by mining our datasets. We can visualize the changing trend of the application energy consumption across multiple versions. Moreover, since system calls sit in the middle of user applications and the OS kernel, it is possible for researchers to model software energy consumption over versions and also trace back to software in order to locate software changes that are responsible for energy consumption variations using system calls.

7. REFERENCES

[1] M. Dong and L. Zhong. Self-Constructive, High-Rate Energy Modeling for Battery-Powered Mobile Systems. MobiSys '11, pages 335–348, USA, 2011. ACM.

[2] S. Godard. SYSSTAT Utilities Home Page. `http://pagesperso-orange.fr/sebastien.godard/`.

[3] A. Gupta, T. Zimmermann, C. Bird, N. Naggapan, T. Bhat, and S. Emran. Detecting Energy Patterns in Software Development . Technical Report MSR-TR-2011-106, Microsoft Research, 2011.

[4] S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. Estimating Mobile Application Energy Consumption using Program Analysis. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 92–101, USA, 2013. IEEE Press.

[5] A. Hindle. Green Mining: A Methodology of Relating Software Change to Power Consumption. In *MSR*, pages 78–87, 2012.

[6] A. Hindle. Green Mining: A Methodology of Relating Software Change and Configuration to Power Consumption. *Empirical Software Engineering*, pages 1–36, 2013.

[7] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-Grained Power Modeling for Smartphones using System Call Tracing. EuroSys '11, pages 153–168, USA, 2011. ACM.

[8] The Linux man-pages project. Linux Man Pages Online. `http://man7.org/linux/man-pages/`, 2013.

[9] C. Zhang. The Impact of User Choice and Software Change on Energy Consumption. Master's thesis, University of Alberta, 2013.

[10] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. CODES/ISSS '10, pages 105–114, USA, 2010. ACM.