# Artifacts for "A Comprehensive Study on the Energy Efficiency of Java's Thread-Safe Collections"

Gustavo Pinto[1] Kenan Liu[2] Fernando Castor[3] Yu David Liu[2]
[1]IFPA, Brazil [2]SUNY Binghamton, USA [3]UFPE, Brazil

## I. Description

### A. Contact

Gustavo Pinto <gustavo.pinto@ifpa.edu.br>
Kenan Liu <kliu20@binghamton.edu>
Fernando Castor <castor@cin.ufpe.br>
Yu David Liu <davidl@cs.binghamton.edu>

### B. License

This artifact is licensed under MIT license.

### C. Summary

The artifacts can be found in the following website: https://www.dropbox.com/sh/1o0twfnfenvgk7m/AAAs35TcpfAMQsX2a8eOyZ8Ja?dl=0

This artifact contains the source code used in the experiments, the `jRAPL`[1] implementation (which helped us to measure energy consumption of the source code used), the raw energy data generated by `jRAPL`, as well as the figures presented in the paper, and the source code used to plot these figures using the raw energy data. Details for using `jRAPL` can be found at the companion website.

## II. Scorecard

### A. Insightful

- *Timely (i.e., addresses a problem that is most current and most pressing)?* This artifact tackles a problem that is becoming increasingly recurring not only in software development community, but also in the agenda of software engineering researchers: energy consumption. In particular, this artifact focus on the usage of different data structures in a multi-core environment. On the one hand, data structures are one of the most important building blocks of computer programming. On the other hand, not only high-end servers but also desktop machines, smartphones, and tablets need concurrent programs to make best use of their multi-core hardware. This artifact makes it easier to reproduce the usage of 16 different data structures two different multicore machines.
- *Makes researchers 'smarter' in some way (e.g., identifies and fills some significant gap in prior work)?* This artifact makes it easier to reproduce the usage of 16 different data structures two different multicore machines. While one can reproduce the artifact by reading the research paper, the artifact greatly reduces the barriers for conducting system-related research focused on data structures. Also, the data structure source code is thoroughly integrated with `jRAPL`, and the `jRAPL` output is thoroughly integrated with the plotting scripts. Therefore, after one get acquainted with the data gathering procedure, the artifact becomes straightforward.

### B. Useful

- *Serves a useful purpose?* Our artifact is useful because it makes our claims reproducible. It was created to better assist energy-aware researchers and developers. It also helped us to publicize a research tool (`jRAPL`) that is being used in recent software engineering studies.
- *Serves a purpose that would otherwise be tedious, prolonged, awkward, or impossible?* With our artifacts (tooling, raw energy data, and plotting scripts), one can easily not only reproduce the main results of our study, but also (1) reuse the artifacts in different energy-aware problems (*e.g.,* with our tool), or (2) automate other system-related studies (*e.g.,* with our scripts). When conducted manually, such activities are time consuming, tedious, and error prone.
- *Cost-effective?* There is no cost related to the usage of the artifacts.

### C. Usable

- *Easy to understand?* Our artifact is well-organized. There is just one Java file for each benchmark used. Also, all Java files employ the same structure, so it is easy to move from one to another. Each Java file generates one CSV file, with the same name but different extension. The same nomenclature also applies for the plotting scripts.
- *Accompanied by tutorial notes?* Each folder accompanies a README file that suggests how one can use the folder.
- *Executable?* The artifact is ready to be executed. However, `jRAPL` needs to be properly installed before used (instructions at the companion website). As a limitation, though, `jRAPL` only works with Intel machines.

[1]http://kliu20.github.io/jRAPL/

# Artifacts for "A Comprehensive Study on the Energy Efficiency of Java's Thread-Safe Collections"

Gustavo Pinto[1] Kenan Liu[2] Fernando Castor[3] Yu David Liu[2]
[1]IFPA, Brazil [2]SUNY Binghamton, USA [3]UFPE, Brazil

## I. Overview

Analyzing the energy consumption of application level software is an emerging direction. This artifact makes available all the toolset and raw data needed to reproduce the main findings of our research paper. The artifact consists of:

- The source code of the micro-benchmarks analyzed;
- The source code of the case study used;
- The `jRAPL` tool;
- The raw energy data generated by the `jRAPL` tool with the source code of the experiments;
- The plotting scripts used to create the figures of the paper, based on the raw energy data;

## II. Requirements

Our scripts have been tested in two Linux machines (kernels 3.16.0-0.bpo.4 and 3.0.0-1-amd64). It requires Java 7, Python 2.7, and the `jRAPL` tool. Installing `jRAPL` is straightforward: one just needs to run the `make` command at the root directory. All other dependencies are automatically configured. At this current stage, there is no known errors, only limitation (see below). If new errors arrive, we invite one to open an issue reporting that[1].

## III. Limitations

In order to run the binaries of the study (*e.g.,* the .jar files), one needs to use the same JDK version used. In this study, the source code was compiled using Java 7. The other limitations of this artifact is related to the tool used to gather energy consumption data: `jRAPL`. `jRAPL` needs super user permissions to access the Machine-Specific Registers (MSR). Such MSRs can be accessed by OS, such as the `msr` kernel module in Linux. However, Windows and OS machines cannot be satisfy this requirement, since they do not provide the `msr` module. Another limitation of the `jRAPL` tool is that it only works for at most 2 sockets CPU. Since most of the commodities hardware available do not use more than 2 sockets, we believe that this limitation will not severe problems.

## IV. Organization and Utilization

The artifact is organized in three main folders. Folder *data* presents the raw energy data for the microbenchmarks. This folder has two other folders: *node6e* and *gaia*. These folders represent System#1 and System#2, respectively. The data for each system is presented inside the particular folder. Each benchmark data is presented separately in a CVS file. Specifically, List data is available in the Lists.csv file, Set data is available in the Sets.csv file, and Map data is available in the Hashs.csv file. Along with the CSV files, there is also a Python code that uses the CSV files to generate the figures available in the paper. We warn users to pay particular attention for the `charter.py` file, which contains the absolute path for the CSV files. However, this path is hard-coded. Therefore, one needs to update the path accordingly.

Folder *microbench* presents the source code for the microbenchmarks, as well as the `jRAPL` implementation. The microbenchmarks are available as Java classes. Similarly to the CSV files, the name of the Java classes are related to the benchmark in use. For instance, `ListTest.java` performs with Lists, whereas `SetTest.java` performs with Sets, and `HashingTest.java` performs with Maps. Additional Java files are available, such as `HashCollisionTest.java`. The `jRAPL` tool is available by means of the C files (*e.g.,* `arch_spec.c CPUScaler.c` files) as well as the Java files inside the *energy*. To run the `jRAPL`, one just needs to execute the `EnergyCheckUtils.java` file. However, `jRAPL` must be installed prior utilization. The `jRAPL` companion website describes how to install it[2].

Finally, folder *case-study* presents the source code for the case study, as well as the raw energy data, and the plotting scripts used to generate the case study figures. The case study is formed by two real world applications: tomcat and xalan. These two benchmarks have their own folder, with the source code and the jar files. In this case, the `jRAPL` library is embedded inside the jar file, so there is no need to configure the tool outside the folder. To run the jar file, this sintax is used: `java -jar jar-file.jar`, where "jar-file.jar" should be replaced by the existing jar file.

[1]https://github.com/kliu20/jRAPL

[2]http://kliu20.github.io/jRAPL/