Supporting the Discovery of Energy Hot-Spots via Performance Alignment

Max Weber

Abstract

Short summary.

Contents

Chapter 1

Introduction

1.1 Motivation

Energy consumption and energy efficiency are big challenges this time. Many areas like economy and politic discuss about it. Countries give themselves thresholds for greenhouse gas output. To achieve these goals they invest in green energy and publish guidelines.

1.2 Contribution

This work tries to help with the detection of energy critical parts of software. Performance analysis of software projects Performance and energy alignment Development of a Eclipse-Plug-in to visualize performance/energy hot-spots

Chapter 2

Background and Related Work

2.1 Background

2.2 Related Work

A Survey about Performance Evaluation of Component-based Software Systems [5].
A Survey about Model-based performance prediction in software development [2].
Cycle-accurate simulation of energy consumption in embedded systems. Comparison between performance and energy computed by their simulator [6].
Report on experience in using the Running Average Power Limit (RAPL) energy sensors available in recent Intel CPUs for measuring energy consumption of short code paths [4].
Early Performance Testing of Distributed Software Applications [3].
Green Tracker measurement tool (for estimating the energy consumption of software)[1].
Software performance engineering [7].
The Future of Software Performance Engineering [8].

Chapter 3

Java Performance Measurement

Measurement of performance is an important aspect of all kinds of science[]. performance measurement of programming languages performance measurement of java

3.1 Overview of Java Performance Measurement Tools

Because of the thing that it is importatant to know how well your program performs many different tools arose in the area of performance measurement. Some of them are developt for specific requirements others are restricted in different ways. In the following part of work there will be a summary of all important measurement tools for Java programs.

3.1.1 HPROF

The Java 2 Platform Standard Edition (J2SE) has always provided a simple command line profiling tool called HPROF for heap and cpu profiling. HPROF is actually a JVM native agent library which is dynamically loaded through a command line option, at JVM startup, and becomes part of the JVM process. By supplying HPROF options at startup, users can request various types of heap and/or cpu profiling features from HPROF. The data generated can be in textual or binary format, and can be used to track down and isolate performance problems involving memory usage and inefficient code. The binary format file from HPROF can be used with tools such as jhat to browse the allocated objects in the heap.
In J2SE Version 5.0, HPROF has been implemented on the new Java Virtual Machine Tool Interface.

3.1.2 NetBeans Profiler

NetBeans profiler is a fully featured Java profiling tool integrated into the NetBeans IDE. The features include CPU, memory, threads, locks and SQL queries profiling as well as basic JVM monitoring, allowing developers to be more productive in solving performance and memory issues.

3.1.3 JProfiler

JProfiler is a commercially licensed Java profiling tool developed by ej-technologies GmbH, and is mainly designed for use with Java EE and Java SE applications. It combines CPU, Memory and Thread profiling into one application and is useful for developers as it can be used to analyze performance bottlenecks, memory leaks, CPU loads and resolve threading issues and supports local profiling (analysis of applications that are running on the same machine on which the JProfiler software is installed) and remote profiling (this is where it allows for the analysis of Java applications which are running on remote machines which JProfiler software is not installed on.).

3.1.4 VisualVM

VisualVM is a tool derived from the NetBeans platform and its architecture is modular in design meaning its easy to extend through the use of plugins.
Visual VM allows you to get detailed information about your Java applications while they are running on a Java Virtual Machine (JVM). Data generated can be generated and retrieved by the Java Development Kit (JDK) tools and all the data and information on multiple Java Applications can be viewed quickly both local and remote running applications. It is possible to also save and capture the data about the JVM software and save the data to the local system, and then view the data later or share it with others.
Visual VM can do CPU Profiling, memory Profiling, run garbage collections, take snapshots and more.

3.1.5 Java Performance Analysis Tool (Patty)

The "Patty" project is aimed at providing a profiling tool for the Java 1.5.0 and higher Virtual Machines only (depending on backwards compatibility in JVMTI interface). The difference with other profilers is this project maintains a very high emphasis on targeted profiling and allows users to switch profiling features on and off at runtime.

3.1.6 Java Interactive Profiler

JIP is a code profiling tool. It allows you to turn the profiler on and off while the JVM is running. JIP is pure Java. It takes advantage of the Java5 feature which allows you to hook the classloader. JIP adds aspects to every method of every class that you want to profile. These aspects allow it to capture performance

data. JIP actually tracks the amount of time used to gather performance data and factors that time out of its analysis. Filters by package/class name.

3.1.7 Profiler4J

Profiler4j is a dedicated CPU profiler Java that is user friendly and supports remote profiling and can be configured on the fly. Notable features include that its based on dynamic bytecode instrumentation, it as no native library nor requires an executable. Further notable features are that its done 100% in Java, can provide graphical information with a call graph, call tree, memory monitor, and class list. and supports fine-grained configuration. It is currently released under the Apache License v2.0.

3.1.8 JRat

The Java Runtime Analysis Toolkit is a low overhead, easy to use, open source performance profiler for the Java platform. JRat monitors an application's execution and persists performance measurements. This data can then be viewed and analyzed using the JRat Desktop, a Swing application.

3.1.9 Extensible Java Profiler

Extensible Java Profiler (EJP) is an open-source profiling tool for Java with a scalable and extensible architecture, allowing its usage for exotic programming languages that use a Java backend.
EJP is based on the Java Virtual Machine Profiler Interface (JVMPI). On the contrary of Sun's hprof tool, which generates statistical information, it logs every single method invocation. It can be used to trace the execution of small parts of Java programs and display it in hierarchical trees with some elements hidden or highlighted.

3.1.10 JMP - Java Memory Profiler

JMP is a profiler for java that can be used to trace objects usage and method timings. Since jmp use jvmpi it only works with java/1.2 up to java/1.5. Work has started on tijmp, a new profiler that uses the the tools interface (jvmti). JMP normally uses one window to show the classes in memory. Each class has summary information for number of instances and total bytes of used memory for all the instances. JMP can perform heap analysis and has the ability to show which objects own (have references to) all the objects of a specified class. This is a great way to find memory leaks. JMP also shows method timings and calls in another window. Several columns show time taken in the method, number of calls to each method, time taken in methods called. JMP collects information about which method are called and from where, this information is used to build call graphs. JMP interacts with the normal java threads and

also uses one extra thread for GTK+ with a timer to systematically update the stats. JMP is written in C, it is designed for speed.

3.1.11 TIJMP - Tools Interface Java Memory Profiler

TIJmp is a memory profiler for java. TIJmp is made for java/6 and later. TIJmp is written to be fast and have a small footprint, both memory- and cpu-wise. This means that the jvm will run at almost full speed, until you use tijmp to find some information. TIJjmp uses C code to talk to the jvm and it uses swing to show the the tables of information. So tijmp is written in C (using jvmti and jni) and Java. TIJmp runs in the same jvm as the program being profiled. This means that it can easily get access to all things jvmti/jni has to offer.

3.1.12 JMeasurement

JMeasurement is a free and simple java api for monitoring runtime and usage (count, parallel activation, last activation) of user defined points in java production code. It is simple to use and extended. JMX is supported. Actual version is always in maven central.

3.1.13 DJProf

DJProf is an experimental tool for profiling Java programs which employs AspectJ to insert the necessary instrumentation for profiling rather than, for example, the Java Machine Profiler Interface (JVMPI). DJProf can be used to profile Java programs without modification (i.e. there is no need to recompile them for profiling) and does not require the user to have any knowledge of AspectJ.

3.1.14 EurekaJ -> Montric

EurekaJ is an Open Source, Standards based profiler tool for Java applications. EurekaJ integrates with the Java agent BTrace. BTrace has a large feature set that goes hand in hand with features for EurekaJ. Motric 1.0 is the further developed version of EurekaJ.

3.1.15 jvmtop

jvmtop is a lightweight console application to monitor all accessible, running jvms on a machine. In a top-like manner, it displays JVM internal metrics (e.g. memory information) of running java processes. jvmtop does also include a CPU console profiler.

3.1.16 Oracle JRockit Mission Control

The JRockit Mission Control tools suite includes tools to monitor, manage, profile, and eliminate memory leaks in your Java application without introducing the performance overhead normally associated with tools of this type.

Bibliography

[1] Nadine Amsel and Bill Tomlinson. Green tracker: a tool for estimating the energy consumption of software. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pages 3337–3342. ACM, 2010.

[2] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004.

[3] Giovanni Denaro, Andrea Polini, and Wolfgang Emmerich. Early performance testing of distributed software applications. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 94–103. ACM, 2004.

[4] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. Measuring energy consumption for short code paths using rapl. *ACM SIGMETRICS Performance Evaluation Review*, 40(3):13–17, 2012.

[5] Heiko Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):634–658, 2010.

[6] Tajana Simunic, Luca Benini, and Giovanni De Micheli. Cycle-accurate simulation of energy consumption in embedded systems. In *Design Automation Conference, 1999. Proceedings. 36th*, pages 867–872. IEEE, 1999.

[7] Connie U Smith. Software performance engineering. In *Performance Evaluation of Computer and Communication Systems*, pages 509–536. Springer, 1993.

[8] Murray Woodside, Greg Franks, and Dorina C Petriu. The future of software performance engineering. In *Future of Software Engineering, 2007. FOSE'07*, pages 171–187. IEEE, 2007.

Acronyms

JVM Java Virtual Machine

Appendix A

Appendix

A.1 Benchmarks