Resource Monitoring for Poly/ML Processes

— Extended Abstract (Demo) —

David Matthews
Prolingua Ltd.
Cambridge, U.K.
david.matthews@prolingua.co.uk

Magnus Stenqvist Tjark Weber
Dept. of Information Technology
Uppsala University, Sweden
magnus.p.stenqvist@gmail.com tjark.weber@it.uu.se

Application monitoring is invaluable to manage the performance of applications, understand their resource requirements and identify bottlenecks. We present a graphical monitoring tool for the Poly/ML run-time system, a popular implementation of Standard ML. This allows to gather and visualize detailed data showing the run-time behavior of Poly/ML applications, and thereby assists developers in understanding and improving the resource needs of such applications.

Software applications are subject to (implicit or explicit) performance requirements. Whether such requirements are met is evident only under load at run-time. Application monitoring is commonly used to understand the computational resources required by software applications, and to identify performance bottlenecks.

Standard ML [6] (SML) is a general-purpose functional programming language. For want of better alternatives, developers of SML applications typically resort to generic system monitor tools, such as the Windows Task Manager [1] or htop on Linux [7], for application monitoring. While these tools provide basic information about a process, they are unaware of the internals of the run-time system that is executing SML code. In contrast, run-time aware monitoring tools such as JConsole [3] for the Java Virtual Machine can provide much more specific information, e.g., about threads and garbage collection.

This extended abstract presents a graphical monitoring tool for the Poly/ML [5] run-time system. Poly/ML is a popular open-source implementation of SML. It features multi-threading and parallel garbage collection, and since version 5.5, it makes detailed information about the state of the run-time system available. Our monitoring tool continually retrieves this data for running Poly/ML processes, and presents it to the user in visual form.

The resource data that is collected by the Poly/ML run-time system currently includes 17 values from four areas of interest: *CPU time* (four values: both user and system time, separately for garbage collection and non-GC), *memory footprint* (five values: total heap size, free space after the last partial/full garbage collection, size of and free space in the allocation area), *garbage collection* (two values: number of partial and full GCs) and *threads* (six values: total number of threads, threads running ML code, threads waiting for I/O, a mutex or a condition variable, threads performing signal handling). Additionally, up to eight *user counters* are available and may be used freely by applications to keep track of integer values: for instance, to aid in monitoring an application's task queues.

This information is exposed via a memory-mapped file whose name is derived from the process identifier of the Poly/ML process. Details differ between Windows and Linux. In earlier versions of Poly/ML, the file was directly mapped from a C struct in memory, making its precise layout subject to the C compiler and other unknowns. In current versions, to facilitate interoperability between processes, ASN.1 Basic Encoding [2] is employed to obtain a well-specified binary format.

Additionally, two functions are provided by the Poly/ML run-time that allow the data to be accessed conveniently from ML applications. `PolyML.Statistics.getRemoteStats` takes the process identifier of a running Poly/ML process and returns an SML record containing the 17 values (plus eight user

Submitted to:
ACM ML Workshop 2015

counters) described earlier. `PolyML.Statistics.getLocalStats` is a simplified version that returns the data for the current process, without requiring a process identifier.

With so much data at our disposal, the challenge becomes to prepare and display this data in an accessible fashion. To this end, we have implemented a graphical monitoring application. The monitor is largely written in Java for easy programming access to GUI components, and uses the JFreeChart library [4] to display multiple time series. Figure 1 shows a screenshot of the main window. The source code of the application, which is structured according to the common model-view-controller pattern, is available from `https://bitbucket.org/tjark/poly-ml-monitor`.

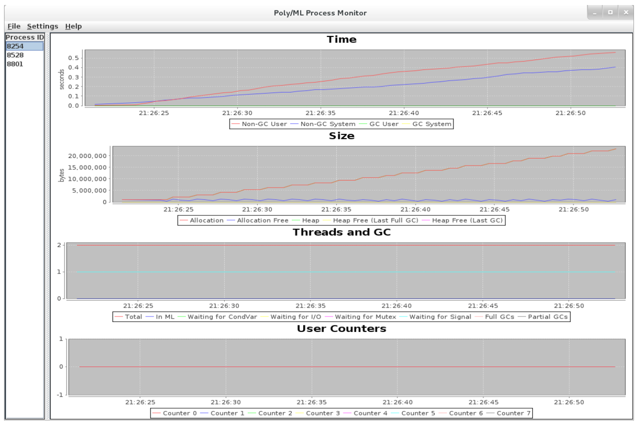

Figure 1: Poly/ML Process Monitor

Since the ML interface to the statistics data, i.e., `PolyML.Statistics.getRemoteStats`, has proved more stable over time than the precise format of the memory-mapped file, the monitor uses the former to retrieve data for all running Poly/ML applications. It launches a small Poly/ML application that receives process identifiers and responds with XML-encoded statistics data via standard pipes. This application is queried for updated values at fixed intervals (every 500 ms by default).

The ML code uses extensible records, so that the monitor application will likely be forward-compatible with future versions of the Poly/ML run-time even if these add further items to the data provided. Of course, actually displaying such additional items in the monitor's UI would require at least minor changes to its sources.

Thus, our monitoring application allows developers to visualize the resource requirements of their Poly/ML applications at run-time. It will run on the most popular architectures and operating systems (provided Java is available), and it does not require any changes to the applications that are being monitored.

References

[1] *How to use and troubleshoot issues with Windows Task Manager*. Available at `https://support.microsoft.com/kb/323527`. Last Review: September 11, 2011. Retrieved May 18, 2015.

[2] *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. Available at `http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf`. ITU-T X6.90, 07/2002. Retrieved May 18, 2015.

[3] *The Java Monitoring and Management Console (jconsole)*. Available at `http://openjdk.java.net/tools/svc/jconsole/`. Retrieved May 18, 2015.

[4] David Gilbert: *JFreeChart*. Available at `http://www.jfree.org/jfreechart/`. Retrieved May 18, 2015.

[5] David Matthews: *Poly/ML*. Available at `http://www.polyml.org/`. Last updated: 22 August 2014. Retrieved May 18, 2015.

[6] Robin Milner, Mads Tofte, Robert Harper & David MacQueen (1997): *The Definition of Standard ML – Revised*. MIT Press.

[7] Hisham Muhammad: *htop - an interactive process viewer for Linux*. Available at `http://hisham.hm/htop/`. Retrieved May 18, 2015.