Tracing methodologies and tools for AI and Data Mining JAVA frameworks
Second Report

Stagi, Roberto
Supervisors: Gil Gomez, Marisa (UPC); Garza, Paolo (PoliTO)

· Universitat Politècnica de Catalunya · Politecnico di Torino ·
· Barcelona Supercomputing Center ·

Abstract. Tracing is an important tool in multi-threaded applications. The tools from the BSC were not designed to be suitable with complex JAVA applications. In this study, these tools have been upgraded to trace more kinds of events from the JVM, and they are applied to some known AI and Data Mining frameworks. The application is focused on revealing some key events to better understand the behaviour of such applications.

1 Introduction

This is the second report on the Thesis in object. The knowledge of the previous report is essential to understand the basis of the work, as well as how to use the examples contained in the repository. Therefore, it is hereby assumed that the reader is aware of its content.

2 What has been done

In this part of the work, the point which has been deepen is the one related to the tracing of the missing events. To do that, two solutions have been considered.

2.1 Using JVMTI Bytecode Instrumentation feature

As explained in the first Report, JVMTI (JVM Tool Interface) is being employed to trace the traces and their basic states (running or blocked in waiting). In order to trace the other interesting functions, one option worth to try was the feature for Bytecode Instrumentation, exposed by JVMTI.

Even if it can be used for very simple tracing programs, it resulted to be extremely difficult to apply to this case. Even if it would have been totally implemented in C, it required the insertion of JAVA instructions. This solution was both difficult to implement and inconvenient.

Difficult because of the lack of frameworks for the Bytecode manipulation available in C (there are some available in C++, but I could not find any available in standard C).

Inconvenient because the requirement of inserting JAVA instructions, would have meant to insert new *native methods*, which needed to be implemented through *JNI* again in C. This whole procedure would be expensive, counter intuitive and pointless.

Moreover, such solution would lack of some very important features that, instead, are given by the actual solution employed and explained in the next paragraph: AspectJ.

The missing features include, but are not limited to:

- ◇ prevention mechanisms in case of faults, errors, exceptions (the tracing may intercept the beginning of a function, but no the end of it)
- ◇ filtering mechanisms on the chosen methods

2.2 Using AspectJ

Using AspectJ, it was possible to trace any kind of function. The problem became to make this usage transparent to the user. An AspectJ file with all the *aspects* is installed during Extrae installation, and it is recalled during the execution of `extraej` (the program to trace Java applications).

AspectJ was already being used to trace the *user functions*. In that case, the file containing the aspects was generated automatically for every different application, by reading a file containing all the names of the functions. In this case, instead, the file is already coded, it is retrieved by `extraej` and used together with the aspects for the user functions.

Using this method, it was possible to intercept the functions in order to trace the events. However, the inter-thread communication tracing methodology, that was already present in the Java tracing probes, is not working. It seems to be lost somewhere, before its display on Paraver. Since I've seen how easy it is to inject new events with these new tools at my disposal, I'm currently focusing on this issue. I haven't found a solution so far, so I'll probably re-implement it from scratch, even if it may imply some complex workarounds.

As it has been already expressed, AspectJ is the solution that it's currently being used to trace the new events. The ease in implementation and maintainability comes at a small cost: the user who intends to trace those events using Extrae, needs to install AspectJ.

2.3 What has not been covered yet

From the last report, it was said that the new changes had to be tested against some real-world applications (even multi-process and "faulty" applications).

This work has not been done yet, because the tracing methods are not complete.

3 What to do from now

From now, I believe that the part involving Extrae and the probes injected in the code is almost over. The infrastructure is almost complete and there are just some small problems to be solved. In my opinion, the next steps need to be:

- ◇ Definitively solve the inter-thread communication issue
- ◇ Adding all the missing events to the ones traced by AspectJ
- ◇ Find the applications to test these changes with
- ◇ Re-iterate