TRACING A JAVA PROGRAM ON JAVA VIRTUAL MACHINE

Karthika Nanthakumar, Tristan M. Basa, Gerhard Dueck
University of New Brunswick
Faculty of Computer Science
knanthak@unb.ca, tristan.basa@gmail.com, gdueck@unb.ca

INTRODUCTION

• Automatic Memory Management (AMM): The Java Virtual Machine (JVM) feature responsible for the following memory-handling tasks: object memory allocation, identification, and deallocation. [2]

Tracefiles: Due to the deterministic nature of a simulator it needs an input file that it can run multiple times in order to present results. This list of commands is called a tracefile.

• Simulator: A simulation program imitates the operation of JVM on a very abstract level. Therefore this simulator will only be able to create objects and move them virtually in a simulated heap. This allows the fast prototyping of new GC techniques.

• Goal: Generate a realistic memory trace of a Java program by instrumenting the JVM.

• Problem: If the tracefile is created artificially, then this may not reflect the behavior of a Java program.

• Idea: Perform trace dumps whenever object allocation, reassignment, and deallocation occurs in the JVM.

BACKGROUND

The configuration of the artificial tracefile is given below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rootsetSize</td>
<td>Size of rootset per thread</td>
<td>7</td>
</tr>
<tr>
<td>threadsNumber</td>
<td>Number of threads</td>
<td>5</td>
</tr>
<tr>
<td>ratioAllocationSetpointer</td>
<td>Number of allocation operations</td>
<td>70</td>
</tr>
<tr>
<td>ratioNullpointerAssignment</td>
<td>Probability of setting a reference to NULL instead an object</td>
<td>20</td>
</tr>
<tr>
<td>maximumPointers</td>
<td>maximum reference slots</td>
<td>10</td>
</tr>
<tr>
<td>minimumPayload</td>
<td>minimum size for an object</td>
<td>1</td>
</tr>
<tr>
<td>maximumPayload</td>
<td>maximum size for an object</td>
<td>64</td>
</tr>
</tbody>
</table>

Three events in the JVM are relevant in creating the tracefile:

1. Capturing Object Allocation: In JVM, object allocation is performed under two functions. Both function calls must be caught in order to dump needed information for the tracefile.

2. Capturing Change in Object Reference: JVM has read and write barriers that intercept the execution of an application. Change in object reference are tracked in these barriers.

3. Rootset Dump: In order to perform a root set dump, two steps are needed: stop-the-world and root set source scanning. Stop-the-world (only the root scanning thread will be running) can only be performed when the VM is at a safe point. This is achieved by signalling the event AsyncEvent in the JVM. This is needed since thread stack frames keep changing while the threads are running.

TRACEFILE FORMAT

The tracefile has the following instructions:

1. a Ti Oj Ss Nn
   -- Allocate an object with the following characteristics:
   • i is the thread number;
   • j is the object id;
   • s is the payload of the object;
   • n is the number of pointers in the object.

2. + Ti Oj
   -- Add object j as a root to thread i

3. - Ti Oj
   -- Remove object j from the root set of thread i

4. r Ti Pm #i Oj
   -- Set the ith pointer of object m to point to object j:

EVALUATION

The figure below shows the statistics between the parameter values of an artificial tracefile (Artificial Trace) and our trace output (Real Trace).

FUTURE WORK

• Implement a dynamic rootset. Currently, a maximum size for the rootset is chosen. In the JVM however, the size of the rootset varies dramatically. Having a dynamic rootset size will make the system more efficient in terms of memory usage.

• Consider information relevant only to the target Java application. Many memory management operations being performed are internal to JVM. Ignoring these operations would result in fewer dumps, and thus a more compact tracefile.