ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 4, April 2014

Detecting Resource Leaks in Java Program

Kiran Deshmukh[1], Aarti Bhagure[2] Prof.Zafar Ul Hasan3 Prof.Anil R.Auti4 Prof .Yogesh R Nagargoje[5]

Student,CSE Department, Savitribai Phule Womens Engineering College, Aurangabad,India 1

Student,CSE Department, Savitribai Phule Womens Engineering College, Aurangabad,India 2,

Assistant Professor, Department of Computer, SITRC Nashik, India3

Assistant Professor ,Department of Computer, Dr. Seema Quadri Institute of Tech, Aurangabad, India 4

Assistant Professor,CSE Department, Savitribai Phule Womens Engineering College, Aurangabad,India 5

Abstract: We present and evaluate a new technique for detecting resource leaks in programs with dynamic memory allocation. A resource leak refers to a type of resource consumption in which the program cannot release resources it has acquired. Typically the result of a bug, common resource issues, such as memory leaks, often only cause problems in very specific situations or after extensive use of an application. To verify the effectiveness of the patterns, experiments are given to use them to detect real resource leaks in large open source projects. After you check your program for threading and memory errors and it is clean, if you still have an intermittent failure that you cannot quite track down, it could be caused by a resource leak.

Keywords: Such as, Detecting Resource Leaks in Java Program, Handling errors.

I. INTRODUCTION

Memory leaks that are, memory allocated but no longer accessible to the programs low program execution by increasing paging, and can cause programs to run out of memory. Detects challenging memory leaks and corruption errors as well as threading data races and deadlock errors.

Unfortunately, it is difficult to manage resources automatically. Java provide efficient garbage collection to aid programmers in managing memory objects, to acquire or release other resources, it still needs to explicitly call APIs provided by third party software libraries or low level systems

These reachable objects survive collections because they may conceivably be used in the course of program execution. While this approach usually works well for reclaiming memory no longer needed by the program, it is a common error on the part of the programmer to leave an inadvertent reference to an object that will never be accessed by the program.

A memory leak is a particular kind of unintentional memory consumption by a computer program where the program fails to release memory when no longer needed. When a program requests OS to allocate some memory by using calls such as 'malloc', 'calloc', new etc; and fails to give back the resource to the OS after using it, then we can say a memory leak has occurred.

A memory leak occurs when object references that are no longer needed are unnecessarily maintained. These leaks are bad. For one, they put unnecessary pressure on your machine as your programs consume more and more resources. To make things worse, detecting these leaks can be difficult: static analysis often struggles to precisely identify these redundant references, and existing leak detection tools track and report fine-grained information about individual objects, producing results that are hard to interpret and lack precision.a single insignificant object can maintain a whole graph of heavy objects in memory. to acquire or release other resources, it still needs to explicitly call APIs provided by third party software libraries or low level systems. For example, the Oracle 9i JDBC Developer's Guide and Reference 1http://www.javaperformancetuning.com/news/news116.shtml warn "If you do not explicitly close your connection and Statement objects, serious memory leaks could occur." In this paper for resource leak handle by JSON parser. Json is short for JavaScript Object Notation, and is a way to store information in an organized, easy-to-access manner**.** In a nutshell, it gives us a human-readable collection of data that we can access in a really logical manner.

II. LITERATURE REVIEW

A key feature of memory management in Java is its garbage-collected heap. A typical garbage collector that comes with Java is a tracing collector, which determines which objects should be preserved in memory by tracing all objects reachable from a set of roots. These reachable objects survive collections because they may conceivably be used in the course of program execution.

In the best case, unnecessary references to individual objects simply degrade program performance by increasing its memory requirements and consequently the collector workload. In the worst case, unnecessary references refer to a growing data structure, parts of which are no longer in use. A recent study1 downtime in production systems was due to resource leaks, including memory leaks, files not closed, locks that aren't released, and so on. For open source

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 4, April 2014

projects, resource leaks can be easily introduced during code check-in. It has been recognized that code refactoring is a leading cause of introducing resource leaks. Careless developers may concentrate on improving functionality while forget to check the resource usage upon check-in. Therefore, it is laborious to review, identify, report, and fix such leaks. For example, it took 23 days and 87 project revisions to fix an IO resource leak, LUCENE-21062, in Apache Lucerne

An example of the heap differencing approach to detect memory leaks in Java is Leakbot [19]. Leakbot combines offline analysis with online diagnosis to find data structures which potentially have memory leaks. The offline analysis takes two heap snapshots and does a complete heap differencing to find parts of the graph which may be leaking. It then identifies the data structure(s) which contains these potential leaks. Feeding this information back into the online system, Leakbot then adds expensive object-instance instrumentation only on those types that have already been identified as potentially leaking. Leakbot requires two program executions, both of which include substantial overheads.

III. ALGORITHM FOR DISCOVERING RESOURCE LEAKS

The Resource leak at each program point is obtained by comparing the results of the two analyses. More precisely:

1. For each method of each class (apart from the main method) we calculate its specifications. The specifications describe the minimal state necessary to
run the method safely.
2. Using the results obtained in the previous step, we calculate the precondition of each subprogram of the main method. Here, the subprogram is defined with respect to the sequential composition. The calculation of the precondition of each subprogram is done in a backwards manner, starting from the last statement in the program. The results are saved in a table as (program location, precondition) pairs.
3. Using the forwards symbolic execution, intermediate states at each program point are calculated and added to the results table computed in step 2.
4. The corresponding states obtained in steps 2 and 3 are compared, and as the preconditions obtained by the backwards analysis are sufficient for safe execution of the program, any excess state that appears in the corresponding precondition obtained by the forwards analysis, is considered a memory leak.

IV. MOTIVATIONAL EXAMPLES

In this section, we give an motivational example to showcase that how we are able to detect an real world
resource leak. We may infer from the code that there exists a resource usage pattern in Cassandra library

1. Connection.createStatement ();
2. Connection.close();

and the missing call to *Connection.close()* is a violation to such a pattern. To extract this pattern, mining must be performed on either Cassandra source codes or its execution traces. However, this patten is not applicable to discover resource leaks in other libraries because the names of classes or methods may vary. The execution trace, and found that *Connection.createStatement()* eventually called *java.io.connection.open()* while *Connection.close()* eventually called *java.io.Connection.close()* in behind. This motivates us to extract the resource usage pattern from standard Java API calls. As these patterns do not rely on any specific library, they are universal enough to discover resource leak on any library that written in Java.

V. RESULT OF RESULT LEAKS

In the experiments, we mine resource usage patterns
on Java IO API(IO) and Java Concurrent API(CONC), which are two subsets of standard Java API. Our mining experiment is conducted on the following scenarios:

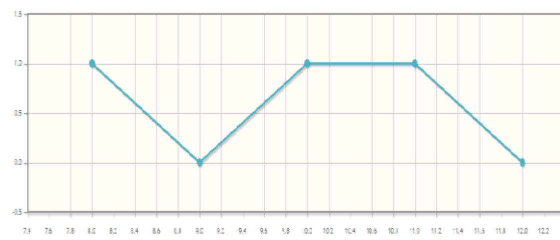

Fig. 1. Program Performance and Result

We can observe from Figure that the number of leaks present in program, the number o patterns, and the number of association rules all decrease with the increase of *min sup*. Besides, the number of patterns would be exponentially large if *min sup* is below 1.0 for IO patterns and 12 for CONC patterns. With the same order of magnitude of running time, the *min sup* of IO patterns is lower than that of CONC patterns, probably 268 because of the multiplicity of Java IO APIs over Java Concurrent APIs. Note that the y-axis of the left figure of Figure is in logarithmic scale.

Fig.2 Total progress of programmer

The above chart shows the total performance of programmer, it define the resource leaks and normal resources which is used in the program.

VI. CONCLUSION

The system provides an detect Resource Leaks in java program. before code check-in, this paper proposes an approach to record the most valueable base API calls during program execution, and mine resource usage patterns from the API call traces. The define programmer progress in graph. we have defined a static analysis algorithm which allows the detection of such allocated and unused objects which cannot be freed by the Resource leaks.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

ACKNOWLEDGMENT

We would like to thank our guide Prof.Y.R.Nagargoje, their guidance and feedback during the course of the project. We would also like to thank our department for giving us the resources and the freedom to pursue this project.

REFERENCES

[1] B. Wright E. Perry, M. Sanko and T. Pfaeffle, "Oracle 9i JDBC developer's guide and reference," Tech. Rep., May 2002.
[2] B. Livshits and T. Zimmermann, "Dynamine: Finding common error patterns by mining software revision histories," in Proceedings of ESEC/FSE, 2005, pp. 296–305.
[3] D. Distefano, P. W. O'Hearn, and H. Yang. A local shape analysis based onseparation logic. In TACAS, pages 287{302, 2006.
[4] D. Distefano and M. J. Parkinson. jstar: towards practical veri_cation for java. In OOPSLA, pages 213{226, 2008.
[5] G. Ammons, R. Bod´ık, and J. R. Larus, "Mining specifications," in Proceedings of POPL, 2002, pp. 4–16.
[6] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das, "Perracotta: mining temporal API rules from imperfect traces," in Proceedings of ICSE, 2006, pp. 282–291.

BIOGRAPHIES

Kiran Deshmukh-I am pursuing degree in computer engineering from Savitribai Phule Womens Engineering College in Aurangabad. My area of interest is java, ASP.net.

Aarti Bhagure-I am pursuing degree in computer engineering from Savitribai Phule Womens Engineering College in Aurangabad. My area of interest is ASP.net, java, DBMS.

Asst.Prof.Zafar UL Hasan is currently working as Assistant Professor, in the Department of Computer, at SITRC Nashik, India3. His research area includes Image Processing,Database,Security

Asst prof.Anil R.Auti is currently working as Assistant Professor in the Department of Computer, Dr. Seema Quadri Institute of Tech, Aurangabad, India .His research area include security,image processing Database .,Soft Computing.

Yogesh Nagargoje is currently working as Assistant Professor in the Department of Computer science and engineering at Savitribai phule women's engineering College, Sharnapur, Aurangabad, Maharashtra, India. He has 3 years of teaching experience. His research area includes Image Processing, Keystroke Biometrics and Mouse Dynamics.