http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (9): 1857-1862, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.1857.1862

Detection and Analysis of Software Aging in a Service-Oriented J2EE Application Server

[1]Meng Haining, [1]Hei Xinhong, [2]Liu Jianjun and [1]Wei Wei
[1]School of Computer Science and Engineering, Xi'an Technology University, Xi'an 710048, China
[2]Aeronautics Computing Technique Research Institute, Xi'an 710068, China

Abstract: Long-running software system tends to show performance degradation and sudden failures, due to error accumulation or resource exhaustion over time. This phenomenon is usually called software aging. This study presents a memory-related detection method to investigate software aging in a service-oriented J2EE application server running on Java Virtual Machine (JVM). By studying the problem of memory leaks in JVM, the memory analyzer tool referred as Profiler is designed and developed to collect the performance data of JVM heap memory. Finally, the experimental results and statistical analysis of collected data validate the presence of software aging in the application server.

Key words: Software aging, application server, web services, java virtual machine

INTRODUCTION

In the study of software reliability, the phenomenon of software aging has been reported in the long-running software system, in which system performance gradually deteriorates over time and even sudden failures occur (Avritzer and Weyuker, 1997). Software aging phenomenon is closely related to system resource consumptions. It has been observed in transaction processing system, web server and Java virtual machine and so on (Garg *et al*., 1998; Grottke *et al*., 2006; Kourai and Chiba, 2011). In order to attack the phenomenon of software aging, software rejuvenation technique as a preventive maintenance policy was first proposed by Huang *et al*. (1995) to ensure system reliability, to reduce high maintenance cost and breakdown cost and to improve system availability.

Web services are emerging as a standards-based platform for application integration across wide area networks and enterprises. Web services can integrate and collaborate with various applications in a loosely coupled way to achieve business goals. It also decreases the complexity of application connection in order to reduce the cost of maintenance and updating. So it is one of the most promising solutions in web application environment, (Tsalgatidou and Pilioura, 2002). Since deployment of web services based architectures has grown over recent years, it is necessary to provide reliable web services (Zhang *et al*., 2012).

The application server is a kind of independent software system or service program that provides management of computing resources and network communications. It shields the complexity and heterogeneity between the underlying network and the operating system, so that the whole system is transparent to client access. In the area of network computing, application server is the supporting runtime platform of web services and the dynamic characters of web services affect systematic behavior and runtime state of application server. We study a service-oriented J2EE application server to see whether it suffers from software aging. Our final objective is to monitor system resources and analyze the cause of software aging which plays an important role in improving the availability and providing more reliable web services of application server.

In this study, we demonstrate the simple methodology for detecting aging in the service-oriented J2EE application server. Since the application server is running on JVM, memory leaks in the JVM is firstly analyzed. Then a memory analyzer tool referred as Profiler is designed and implemented to collect performance resource usage and system activity data from application server at regular intervals. Finally, the evaluation metric of slope estimation technique in the linear regression method is adopted to estimate the presence of software aging in application server and the cause of software aging is given.

RELATED WORK

Numerous and valuable studies have been devoted to detection and analysis of software aging. Garg *et al*. (1998) first proposed a measurement-based method to estimate software aging, the operating system resource

Corresponding Author: Meng Haining, School of Computer Science and Engineering, Xi'an Technology University, Xi'an 710048, China

usage and system activity data such as memory usage were collected via a SNMP-based distributed monitoring tool and then software aging in UNIX workstation is estimated and verified based on a statistical model. Cassidy *et al.* (2002) explored the feasibility and practicability of exploiting advanced statistical pattern recognition for detecting software aging in large OLTP DBMS servers. To estimate aging trends in a Apache web server, the non-parametric statistical methods and parametric time series models were applied by Grottke *et al.* (2006) which demonstrating a numerical validation based on collecting the data of resource usage and activity parameters such as used swap space, response time, free physical memory. By adopting the aspect-oriented programming technology, Alonso *et al.* (2010) injected the monitoring solution in runtime J2EE applications to determine the component root cause of software aging. Kourai and Chiba (2011) collected and analyzed data of throughput loss and memory depletion and then adopted both parametric and non-parametric statistical techniques to reveal the presence of software aging phenomenon in the Sun Hotspot Java virtual machine. The virtual and resident memory utilization was investigated by Araujo *et al.* (2011) to indicate the presence of software aging in the cloud computing infrastructure. Matos *et al.* (2012) monitored and collected the data of RAM memory exhaustion, swap memory usage and CPU utilization, to analyze the software aging effects on the elastic block storage management of eucalyptus framework.

The statistical analysis method and the resource usage parameters including memory usage monitored in this study are similar to the above studies. In contrast to these studies, various system workloads are considered in this study for detecting software aging. In addition, our investigation focuses on the J2EE-based applications and takes into account memory leaks in JVM to analyze the root cause of software aging.

MEMORY LEAKS IN JVM AND THE MEMORY ANALYZER TOOL

Memory leaks are known to be a major cause of reliability and performance issues in software system and they are often a contributing factor to software aging. Despite the built-in garbage collector in JVM, memory leaks can exhaust available system memory as an application server runs on JVM. Therefore, the JVM memory management mechanism is firstly introduced to analyze the problem of memory leaks.

Memory leaks in JVM: JVM memory region, also called runtime data area, can be divided into the area of method, heap, stack, register and native code stack. While a program is running in JVM, JVM memory region can store data such as byte codes, objects, parameters, return values, local variables and intermediate results and so on. Where, the heap memory is used to store instances of classes or array of a runtime Java program. JVM provides the instruction of allocating a new object, but doesn't support memory release which is implemented by garbage collector. However, garbage collector only reclaims useless and unreferenced object as shown in Fig. 1, so that the useless and reachable objects cannot be released from memory area by garbage collector. Therefore, such potential defects of garbage collection mechanism probably cause memory leaks in JVM.

If the problem of memory leaks can not be resolved, JVM memory usage will continue to increase over time and reach to the maximum memory usage for JVM. At the moment, the garbage collection mechanism automatically starts up to gradually release JVM memory for a period time and JVM will occupy a large amount of CPU time and system resources for garbage collection. Nevertheless, the resource is finite, thus it will eventually result in system crash or software aging.

The memory analyzer tool: In view of memory leaks in JVM, a special tool is required to monitor and detect JVM memory, so that developers can easily estimate whether software aging exists in application server.

Application server provides the bidirectional interface JVMPI (Java Virtual Machine Profiler Interface) between JVM and external programs, as shown in Fig. 2. Agent is loaded when JVM is starting up. Agent communicates with JVM via JVMPI and it can receive various events from JVM and send control information to JVM. The Agent module is realized by local language C. Based on Java's platform independence, JVM can call the

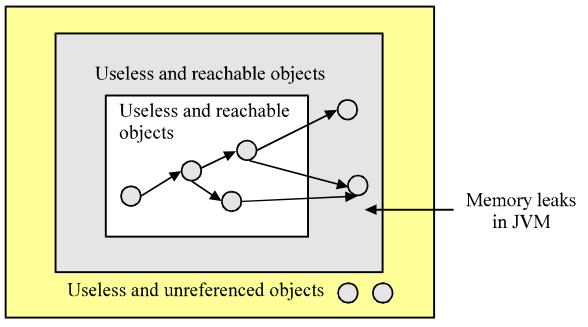

Fig. 1: Memory leaks in JVM

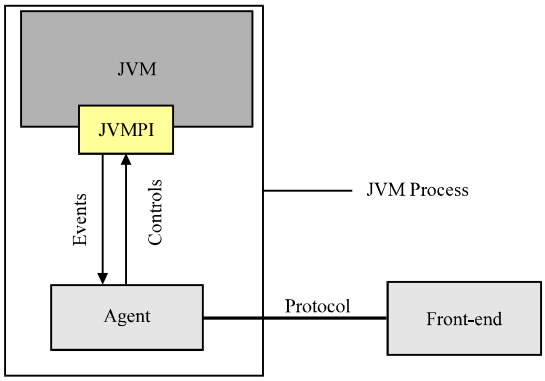

Fig. 2: JVMPI working principle

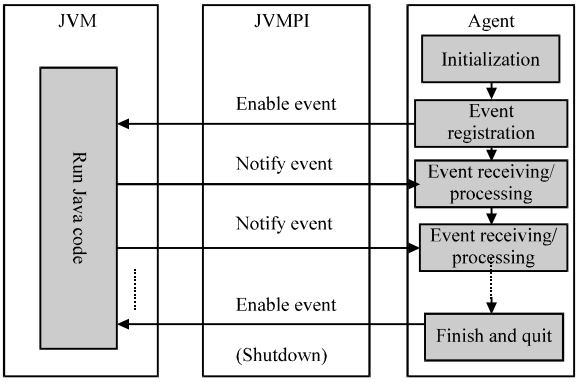

Fig. 3: Event triggering mechanism of the agent

Agent module. In the Agent module, the pointer to an instance of JVM is obtained by using the JNI (Java Native Interface).

The event trigger mechanism is shown in Fig. 3. As the event receiver, Agent registers its interesting events (such as the memory allocation) to JVM which is the event trigger, so that JVM can notify the Agent when the event is triggered.

Based on JVMPI, the memory analyzer tool called Profiler is designed and developed. The main functions of the Profiler include: monitoring memory usage through the visualization window in real-time, selecting sampling frequency of memory data according to the need of users and filtering the output information. The Profiler can collect the data from four modules: JVM runtime module, system data module, web thread module and transaction manager module. JVM runtime module is responsible for extraction of the size of JVM memory and used memory. System data module extracts the CPU usage and free memory usage. Web thread module is in charge of extracting the count of created threads, destroyed threads, concurrent threads. Transaction manager module extracts information of global transactions. Among the four modules, JVM runtime module and system data module are the two most important modules.

AGING DETECTION AND ANALYSIS OF APPLICATION SERVER

The experimental platform simulates a monitoring and recording system for a service-oriented J2EE application server.

Experimental environment: As shown in Fig. 4, the experimental environment consists of a J2EE application server, multi-clients and a database server. In the clients, the load generator is used to generate web service requests to the application server through standards-based HTTP or SOAP protocols. The application server receives the requests, connects and queries the database server and then returns results to the clients. The memory analyzer tool referred as profiler is used to monitor and collect the data of JVM heap memory from the application server.

All the servers involved are 3.0 GHz Pentium IV system running Windows XP, with 1.0 GB of memory. The application server is Websphere 5.1 with maximum JVM heap memory 256 MB and the use case deployed on the application server is Petstore 1.3.1-02. The database server is MySQL. The machines are connected on a same local area network with 100 Mbps Ethernet.

The dynamic parameters in the clients and the application server are periodically monitored and recorded in a certain format separately. The sampling interval is ten minutes. The online access behavior of users obeyed the Poisson distribution in the form of a week period. The load density is different between business days and rest days.

Note that the Profiler runs on the application server and it also occupies part of system resources. Hence, the resources data obtained by this method are in fact the accumulation data of the Profiler and the application server. Nevertheless, the memory analyzer tool Profiler only occupies fewer system resources, so its effect on the aging detection results can be ignored. Thus, the aging detection results can be regarded as the ones of the application server itself.

Peak load test: The peak load of application server is the maximum amount of concurrent client requests that an

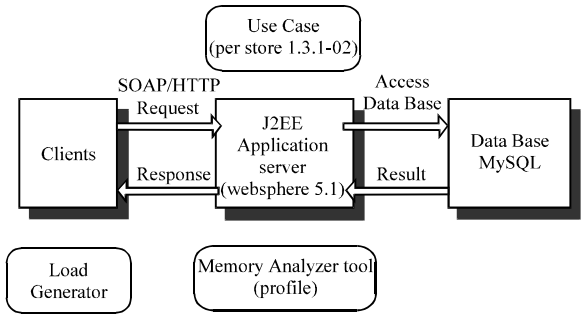

Fig. 4: Software aging detection model for the service-oriented J2EE application server

application server could respond within a certain time period. In order to simulate the peak load, load requests with the equivalent intensity are sent to an application server in a short time. If all these requests are successfully responded by the application server, then the load intensity is increased until there is no response from the application server, when the load intensity is regarded as peak load.

According to the value of peak load, different scenarios with various load intensities are designed to detect and analyze software aging in the application server. Due to the request number per unit time and the mean service time are the main factors influence the load intensity, they are selected as the preconditions in the following aging tests.

Aging test under heavy load: In the heavy load test, the average load intensity is designed to 30% of peak load and the load intensity in the peak period is determined as 50% of peak load. The default initial size of JVM heap memory usage is 128MB and the maximum is 256MB. The application server is unable to respond to client requests after running 43 h, when JVM heap memory usage reached about 250MB.

The relationship diagram between JVM heap memory and running time of application server is shown in Fig. 5. Our practical task is to verify whether software aging exists in the application server, namely, whether the system performance gradually degrades over time. A reasonable approach is to analyze the relationship between JVM heap memory and running time of application server via linear regression method. And the estimated times to JVM heap memory exhaustion was computed using the linear regression equation Y = m*X+c, where m is the slope, c is the intercept or the initial value and Y is the final value (Grottke *et al*., 2006). In this heavy load test, the linear regression equation is fitted as follows:

$$Y = 25.8527*X+137920 \quad (1)$$

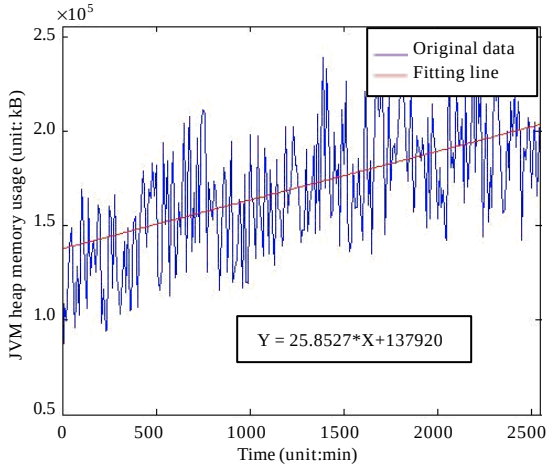

Fig. 5: JVM heap memory usage of application server under a heavy load

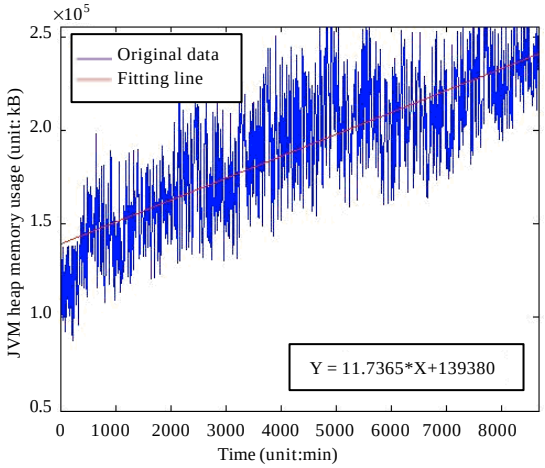

Fig. 6: JVM heap memory usage of application server under a light load

where, X is the running time of application server and Y is the JVM heap memory usage. It can be seen that the slope of regression line in the Eq. 1 is positive which indicates that the JVM heap memory usage increases with the running time of application server. Thereby, it is evident that software aging exists in the application server.

Aging test under light load: In the light load test, the request number in unit time is decreased. The average load intensity and the load intensity in the peak period are being reduced by 20% of the heavy load test. In this test, the application server is unable to respond to client requests after running 140 h. The JVM heap memory of application server is shown in Fig. 6.

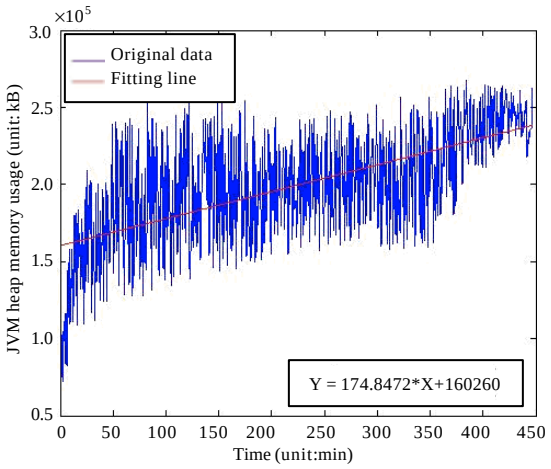

Fig. 7: JVM heap memory usage under prolongation of the mean service time

Similarly, when JVM heap memory usage reached about 250MB, the application server crashed. The linear regression equation which describes the relationship between JVM heap memory and running time of application server is as follows:

$$Y = 11.7365*X+139380 \quad (2)$$

Note that the slope of regression linear Eq. 2 is positive and it is smaller compared with that in the heavy load test. It verifies that software aging exists in the application server and the depletion of JVM heap memory over time is the major cause of software aging. In addition, compared with heavy load test, the load intensity in the light load test is being reduced by 20% yet the running time increases three times. Therefore, it can be concluded that the load intensity is a major factor that influences software aging.

Aging test under prolongation of the mean service time: Next, the mean service time is prolonged six times and the load intensity is equivalent to that of light load test.

The application server crashed after running 43 hours. The JVM heap memory is shown in Fig. 7 and the linear regression equation is as follows:

$$Y = 174.8472*X+160260 \quad (3)$$

Likewise, the slope of regression linear Eq. 3 is positive and it is greater than the ones of the above two kinds of test. This is because of the prolongation of the mean service time, namely, increasing the residence time of users in application server which requires more available system resources. Thus it will greatly accelerate resource consumption and aging process in the application server.

CONCLUSION AND FUTURE WORK

Although the garbage collection mechanism can be regarded as the special Java function to release objects automatically by the garbage collector, the problem of memory leaks still exists in Java applications. And memory leaks are a major cause of software aging. In this study, the aging detection method for the service-oriented J2EE application server is presented and the design and implementation of aging detection are described. Through monitoring the performance parameters of runtime application server, the JVM data is extracted for aging analysis and verification. Finally, through the experimental and statistical analysis of JVM usage, the gradual increase in JVM heap memory usage is visible and the existence of aging is evident. It can be concluded that memory exhaustion is the main cause of software aging in application server and the system workload has great influence on the memory usage. The aging symptoms detected in this study may also occur in other systems based on J2EE architecture.

Future research mainly includes monitoring and analysis of other system parameters of resource consumption. And the study on software aging mechanism is another avenue for new development.

ACKNOWLEDGMENTS

The author would like to thank the sponsors of the National Natural Science Foundation of China under Grant No. 61100173, Scientific Research Plan Project of Shaanxi Education Department of China under Grant No. 09JK642, Doctoral Fund No. 116-210912 and Scientific Research Plan Project of Xi'an Technology University under Grant No. 116-210907.

REFERENCES

Alonso, J., J. Torres, J.L. Berral and R. Gavalda, 2010. J2EE instrumentation for software aging root cause application component determination with AspectJ. Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, Workshops and Ph.D. Forum, April 19-23, 2010, Atlanta, GA., USA., pp: 1-8.

Araujo, J., R. Matos, P. Maciel and R. Matias, 2011. Software aging issues on the eucalyptus cloud computing infrastructure. Proceedings of the IEEE International Conference on System, Man and Cybernetics, October 9-12, 2011, Anchorage, AK., USA., pp: 1411-1416.

Avritzer, A. and E.J. Weyuker, 1997. Monitoring smoothly degrading systems for increased dependability. Empirical Software Eng., 2: 59-77.

Cassidy, K.J., K.C. Gross and A. Malekpour, 2002. Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers. Proceedings of Dependable Systems and Networks, June 23-26, 2002, Washington, DC., USA., pp: 478-482.

Garg, S., A. van Moorsel, K. Vaidyanathan and K.S. Trivedi, 1998. A methodology for detection and estimation of software aging. Proceedings of the 9th International Symposium on Software Reliability Engineering, November 4-7, 1998, Paderborn, Germany, pp: 282-292.

Grottke, M., L. Li, K. Vaidyanathan and K.S. Trivedi, 2006. Analysis of software aging in a web server. IEEE Trans. Reliab., 55: 411-420.

Huang, Y., C. Kintala, N. Kolettis and N. Fulton, 1995. Software rejuvenation: Analysis, module and applications. Proceedings of the 25th International Symposium on Fault Tolerant Computing, Jun 27-30, 1995, Pasadena, CA., USA., pp: 381-390.

Kourai, K. and S. Chiba, 2011. Fast software rejuvenation of virtual machine monitors. IEEE Trans. Dependable Secure Comput., 8: 839-851.

Matos, R., J. Araujo, V. Alves and P. Maciel, 2012. Experimental evaluation of software aging effects in the eucalyptus elastic block storage. Proceedings of the IEEE International Conference on System, Man and Cybernetics, October 14-17, 2012, Seoul, South Korea, pp: 1103-1108.

Tsalgatidou, A. and T. Pilioura, 2002. An overview of standards and related technology in web services. Distrib. Parallel Databases, 12: 135-162.

Zhang, H., H. Chai, W. Zhao, P.M. Melliar-Smith and L.E. Moser, 2012. Trustworthy coordination of web services atomic transactions. IEEE Trans. Parallel Distrib. Syst., 23: 1551-1565.