ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

A Review Survey on Deadlock Detection in Multithreading

A. Mohan[*1], Dr. P. Senthil Kumar[2]

[*1] Research Scholar(Anna University), Saveetha Engineering College, Chennai – 602105, Tamil Nadu, India

[2] Professor, SKR Engineering College, Chennai – 600123, Tamil Nadu, India

mohanmail@sify.com

Abstract

Deadlock freedom is the major challenge in developing multithreading programs. To avoid the potential risk of blocking a program, prior monitoring of threads can be made during the execution process. The proper monitoring scheme can monitor the threads and can identify whether the threads might enter a deadlock stage. It maintains a back up to store the threads. So after the execution of one thread the injection of the other thread can be made from backup into the processing stage. By using this process the deadlock can be avoided in the multithreading environment. In the proposed system, the thread monitoring and thread mapping techniques are implemented to identify the threads running in the program. A map is present which is used to store the thread objects, the locks acquired and requested by them. Whenever a thread tries to acquire a lock and if the access is denied, then it waits for certain period of time. After the time period expires, the thread again tries to access the lock. If the access is still denied then the thread traverses the map to identify the threads that have requested or held the same locks requested by it. If it finds any such threads then it detect that deadlock has occurred. The deadlocked threads wait for each other for infinite time. Now the thread releases all the locks acquired by it, thereby allowing the deadlocked threads to complete their operations. If more than one thread detects deadlock, then priorities are assigned to them at random manner. According to the priorities of threads, they wait for a while (i.e. let other threads to complete their operation). According to the priority, thread execution states are changed. It helps the threads to recover from deadlock situation and allows the threads to complete their execution.

Keywords: Multithreaded program, synchronization, deadlock monitor, thread map, priority.

Introduction

Deadlock-freedom is a major challenge in developing multi-threaded programs, as a deadlock cannot be resolved until one restarts the program (mostly by using manual intervention). To avoid the potential risk of blocking, a program may use try-lock operations rather than lock operations. In this case, if a thread fails to acquire a lock, it can take appropriate action such as releasing existing locks to avoid a deadlock. In the existing system, the usage of mapping is not implemented. In another approach circular mutex wait deadlocks and lock graphs are cleared, but this model is not suited for all environments. The existing dynamic methods have less efficiency when compared to the static deadlock analysis method. The proposed system provides an efficient mapping technique for avoiding deadlocks depending upon priority.

The main aim of the project is to avoid the deadlocks occurred in the threads during execution. It is done by providing a map that stores the thread objects, locks acquired and requested by the thread. In this case, if a thread fails to acquire a lock, it can take appropriate action such as releasing existing locks to avoid a deadlock.

In order to avoid deadlocks in threads during the execution process a monitor is introduced in the proposed work that identifies the threads running in the program i.e. the thread objects are identified. After this process a map is generated that store the thread objects and the locks acquired and requested by them. Whenever a thread tries to acquire a lock and if the access is denied then it waits for certain period of time. After the time period expires, the thread again tries to access the lock. Due to some reasons if accessing the locks is still denied then thread traverses the map to identify the threads that have requested or held the same locks requested by it. If it finds any such threads then it recognizes that deadlock has occurred after which the deadlocked thread will wait for each other for infinite time. When

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

it finds that deadlock condition prevails, the thread releases all the locks acquired by it, so that it might allow the deadlocked threads to complete their required operation. In another scenario if more deadlocks are detected then according to the priority, the execution of the priority based threads are executed in random manner. During this process the threads involved in execution are been backed up. According to the priority the threads execution states are changed. This helps the threads to recover from deadlock situation and let the other threads to complete their execution.

Survey On Deadlock

When developing a dynamic deadlock detection technique for multithreaded programs, a multi disciplinary approach is essential. In the static and dynamic techniques are used for exposing deadlock potential[1]. It has three extensions to the basic algorithm (logic graph) to eliminate and to label as low severity or false warning of possible deadlocks[1,2]. The extension of lock graph algorithm is to detect the deadlock in static and dynamic techniques[1,2].

In, a new technique in practical static race detection is proposed for parallel loops in java[2]. The utilization of these constructs and libraries improves accuracy and scalability[2,1]. The new tool called IteRace has been introduced which includes a set of methods that are specialized to employ the intrinsic thread, safety, and dataflow structure of collections[2]. The IteRace is fast and perfect enough to be realistic. It scales well for programs of lakhs of lines of code and it reports little race warnings, thus shuning a common consequence of static analyses. The tool implementing this method is fast, does not delay the programmer with many warnings, and it finds latest bugs that were conformed and fixed by the developers[2].

In, detecting atomicity violations using dynamic analysis technique is presented[3]. A more fundamental noninterference property is atomicity. When a method execution is not affected by concurrently-executing threads, then that method is called as atomic method[3]. It contains both formal and informal correctness arguments[3,2]. Detecting atomicity violations combine ideas of both lipton's theory of reduction and early dynamic race detectors. It is effective error detection for unintended interactions between threads. It will be more effective than standard race detectors[3,2].

We proposes the type inference algorithm for rccjava. The performance of the algorithm is applied on programs up to 30,000 lines of code. The resulting annotations and race-free guarantee our type inference system. Type inference algorithm is applied to the concurrent program to manipulate the shared variable without synchronization[4]. This algorithm has some lock variables. Extending this inference algorithm to larger benchmark has some issue. It produces reliable error reporting.

They describe an approach for online deadlock detection for multithreaded programs using the prediction of future behavior of threads. 74% of deadlocks were predicted using the proposed method[5]. Some specific behaviors of threads are extracted at run time and converted into predictable format using Time series method. The proposed method has several advantages compared to the existing static methods[5,4]. Powerful technique is used for predicting complex deadlocks[5].

We implemented an efficient algorithm to sense concurrent programming errors online[6]. System programmers monitor the program events where locks are approved or handed back, and in places where values are accessed that may be shared among multiple Java threads[6,5]. The proposed RACER algorithm uses ERACER for memory model of java and AspectBench compiler for implementation. In this paper, they projected a language extension towards the aspect-oriented programming language AspectJ[6]. The proposed AspectJ have implemented the following three points. They are Lock(),Unlock(), Maybeshared().

Examines the performance scaling of various processor cores and application threads. It analyzes the performance and scalability by correlating low-level hardware data to JVM threads and system components[7, 4,3]. It uses the JVM tuning techniques to solve the problems regarding lock conditions and memory access latencies. The study of performance and scalability of multi threaded java application on multi core systems is done. The proposed method reduces the bottlenecks using JVM tuning techniques. Inappropriate use of synchronization leads to large number of stall cycles[7,4,3].

Present a novel dynamic analysis method to find real dead- locks in multi-threaded programs. DEADLOCK- FUZZER is the new technique used to find the deadlocks in two phases[8]. In the First phase, a potential deadlock in a multi-threaded program is found using dynamic analysis technique by execution of the program. In the second phase, deadlock creation is controlled using threads scheduler[8]. DEADLOCK-FUZZER is implemented to find the all previously known deadlocks in large benchmarks, but it does not discover previously

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

unknown deadlocks in an efficient manner. This technique needs both static and dynamic techniques[7,8].

A new Java thread deadlock detection approach called as JDeadlockDetector. This system requires source code and built on non-official JVMs for Java thread deadlock detection solutions[9]. Many Java programs cannot be evaluated with these solutions. JDeadlockDetector is fabricated on the official Java Virtual Machine, viz., OpenJDK's HotSpot. JDeadlockDetector have three unique advantages compared to the existing system[9]. They are application transparency, detection accuracy and minimized performance overhead. JDeadlockDetector attains no false negative and diminished false positive. JDeadlockDetector detects Java thread deadlock based on the capability of monitoring the thread states and synchronization states on runtime[9,10]. In this way the technique achieves their advantages. To track the control flow and data flow of a Java program, Hotspot introspection architecture has to be extended. This will afford a capability to analyze the vulnerability of Java programs[10].

A new two phase deadlock detection scheme was introduced which provides efficient memory utilization and time constraints. The performance of the proposed system is much higher than the traditional approach in finding the potential deadlock in application[11,12]. First phase reduces lock by filtering out certain locks that cannot participate[12]. Second phase creates smaller lock graph for potential deadlock detection. The proposed work can minimize the overall deadlock detection time and increases the performance[12].

We focus on developing dynamic deadlock detection technique which reduces the deadlock occurrences.

Result

The proposed system maintains a deadlock map that holds the information about all the threads running in a program. This will reduces the deadlock occurrences based on the random priority assignment. The deadlock monitor is control the overall process of program execution.

Conclusion

This paper provides an efficient way for accessing shared memory by the multiple threads using deadlock monitor. Deadlock monitor can be used to monitor the thread process regularly. Monitor will reduce the deadlock occurrences. Thread map will be used for dynamic deadlock detection method. Also, it reduces the cost of accessing memory and improves the efficiency of multithreaded applications.

References

[1] *Agarwal R. Bensalem S. et al., (2010) 'Detection of deadlock potentials in multithreaded, programs' IEEE Transactions Vol. 54 No. 5 Paper 3.*

[2] *Cosmin Radoi, Danny Dig (2013) 'Practical Static Race Detection for Java Parallel Loops' In Proceedings of the 2013 International Symposium on Software Testing and Analysis (ISSTA '13) received ACM SIGSOFT Distinguished Paper Award, Lugano, Switzerland.*

[3] *Cormac Flanagana, Stephen N.Freundb (2008) 'Atomizer: A dynamic atomicity checker for multithreaded programs', Elsevier - Science of Computer Programming 71, pp. 89–109.*

[4] *Cormac Flanagana, Stephen N. Freundb (2006) 'Type inference against races' Elsevier -Science of Computer Programming 64, pp.140–165.*

[5] *Elmira Hasanzade Kashan, Seyed Morteza Babamir (2012) 'Artificial Neural Network Based Model for Online Prediction of Potential Deadlock in Multithread Programs' The 16th CSI International Symposium on Artificial Intelligence and Signal Processing, pp. 417-422.*

[6] *Eric Bodden and Klaus Havelund (2010) 'Aspect-Oriented Race Detection in Java' IEEE Transactions on Software Engineering Vol. 36, No. 4, pp. 509-527.*

[7] *Kuo-Yi Chen, J. Morris Chang and Ting-Wei Hou (2011) 'Multithreading in Java: Performance and Scalability on Multicore Systems' IEEE transactions on computers Vol. 60, No. 11, pp. 1521-1534.*

[8] *Pallavi Joshi, Chang-Seo Park, Koushik Sen and Mayur Naik (2009) 'A Randomized Dynamic Program Analysis Technique for Detecting Real Deadlocks', ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'09), Dublin, Ireland.*

[9] *Williams A. Thies W. and Ernst M. D. (2005) 'Static deadlock detection for java libraries' In Proceedings of the 19th European conference on Object-Oriented Programming, ser. ECOOP'05. Berlin, ,Heidelberg: Springer-Verlag, pp. 602–629.*

[10] *Wang Y. Lafortune S. Kelly T. Kudlur M. and Mahlke S. (2009) 'The theory of deadlock avoidance via discrete control' In Proceedings of the 36th annual ACM SIGPLAN-SIGACT*

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

symposium on Principles of programming languages, POPL '09. New York, NY, USA: ACM, pp. 252–263..

[11] Yan Wen , Jinjing Zhao et al., (2011) 'Towards Detecting Thread Deadlock in Java Programs with JVM Introspection' International Joint Conference of IEEE TrustCom-11,pp. 1600-1607.

[12] Zhi Da Luo, Raja Das, Yao Qi (2011) 'MulticoreSDK: A Practical and Efficient Deadlock Detector for Real-world Applications' IEEE international conference on software testing, Verification and Validation, pp. 309-318.

[13] Jula H. Tralamazza D. Zamfir C. and Candea G. (2008) 'Deadlock immunity: enabling systems to defend against deadlocks' In Proceedings of the 8th USENIX conference on Operating systems design and implementation, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, pp. 295–308.