동시 시스템에서의 관점 지향 프로그래밍 기반 프로세스 동기화

아사드아바스 O, 이스마파라시디키, 김가연, 이욱진
한양대학교 컴퓨터공학과
asadabbas@hanyang.ac.kr, isma2012@hanyang.ac.kr, ruredi@hanyang.ac.kr, scottlee@hanyang.ac.kr

Aspect-Oriented Programing based Process Synchronization in Concurrent Systems

Asad Abbas O, Isma Farah Siddiqui, Gayeon Kim, Scott Uk-Jin Lee
Department of Computer Science & Engineering, Hanyang University

ABSTRACT

In multithreading software, avoidance of the race condition is obligatory for nondeterminism in parallel processing. In concurrent systems where many threads compete to access the shared resources, the race condition occurs. In Lock-based programing for complex multithreaded system, the high occurrence of deadlock between threads is still an ongoing problem in research industry. In this paper, Aspect Oriented Programing (AOP) based methodology is presented to get the synchronized processing very effectively with avoidance of race condition and deadlock. In AOP, Synchronized Block Join Point (SBJP) has capability to combine the join points for synchronization of process actions.

1. INTRODUCTION

Multiple threads in multithreading program performs faster to achieve required jobs than sequential programming. Multithreading face so many problems due to inherent non-determinism in scheduling of threads which is very hard to detect and fix. Synchronization of threads is very important to make the multithreaded program deterministic [1]. Due to race condition, Non-determinism of thread execution happens. Race condition among threads can be handled by adding the lock at synchronized block, but it make the program scope very limited with high complexity. Non-deterministic defects occurs because of incorrect orders of memory accesses initiated by the incorrect practice of parallel program-language constructs [2]. Multithreaded system can be simplified by enforcing execution determinism. In the presence of data races in parallel systems do not ensure the determinism of threads or processes [3].

AOP supports the modularization of aspects and crosscutting concerns. It is an application which indirectly invoke the aspect functionality. By using aspect advices, program becomes comprehensive and more sophisticated. Advice of AOP language is mostly considered as base part of the program. Advice consist of pointcuts and joint-point which can be applied at any segment of a program with different constraints and synchronization methods. The execution behavior of an aspect is defined by an advice which is applied at multiple targeted join-points. The execution of advice in program location is identified by join-points and pointcuts. In AOP system, all pointcuts are called automatically with join-point as sequentially [4].

In this paper we discuss the implemented action of SBJP by using AOP. In this paper we also present an approach to get determinism and synchronization in multithreaded programing by using AOP.

The rest of this paper is organized as follow. Section 2 presents related work. Section 3 presents problems of concurrency in multithreaded system. Section 4 discuss the SBJP. Section 5 present our proposed approach. Finally, Section 6 gives conclusion and future work.

2. RELATED WORK

Marques et al. discussed testing of multithreaded application. Authors identified that it is very hard to detect and trace the concurrency and non-deterministic problems of threads [5]. The Cooperari framework has been discussed and it defined different

interception points using Java. By using aspect pointcuts, trace of non-determinism is much beneficial [5].

Jörg Kienzle et al. discussed failure of concurrency in distributed system. Authors discussed the positive points for using AOP for code factorization. It is much beneficial to use AOP in order to avoid concurrency problem in distributed system [6].

Babamir et al. discussed thread competition and possibility of deadlock because of shared resources. These errors are raised because of specific schedules concurrent processing of threads. To resolve the problem AOP was used to target code automatically [7].

Previous works present the SB to get synchronization of processes. In our work we present AOP advices and pointcuts to get the determinism of synchronized block in order to avoid race-condition and deadlock.

3. PROBLEMS In CONCURRENT SYSTEM

Starvation or deadlock is most common problem in concurrent programs when multiple threads compete with each other to access some shared resource. To avoid such problems, locking mechanisms have to be applied on shared resources with some constraints of read and write actions. Deadlock occurs when many threads are waiting to access shared resources which are hold by another threads and Vic-versa. Deadlock depends on workload, timings and compiler options.

4. SYNCHRONIZED BLOCK JOIN-POINT

In Java-multithreading "synchronized" is keyword for critical section to handle the race condition of threads. "Synchronized block" holds the critical section of system to get the synchronization access and avoid the race condition of threads. Synchronized block is not easy to handle the critical section or mutex. Mutual exclusion is not only the element which should be handled but visibility and ordering of threads is also important elements in multithreading.

SBJP contains the behavior to recognize the characteristics of synchronization method. To get the scheduling and visibility of threads in Java-multithreading, AOP offers Join-points to execute synchronized block (critical section) according to advice definitions of execution. AOP advices on different join-points on synchronized block to make sure the visibility and ordering of threads.

In SBJP, the invocation of two or more synchronized methods is not possible at same time because of join-point and advices. During the execution of synchronized block under one thread, all threads will wait until first thread leaves the synchronized block according to definition of advices.

5. AOP-BASED SYNCHRONIZATION

This section discuss our proposed methodology for supporting the aspect advices to get synchronization of methods at various points.

We introduce AOP concepts in Java-multithreading in pointcut determines the location where the aspect has to execute. Advice is a guideline to determine when the certain method or process has to execute i.e. before, after, around or within of a synchronized block. Example below shows a synchronized block and some advices which can make this block deterministic.

Example:

```
public void execute() {
        @before("execution(public String org.koushik.get() )")
         synchronized block() {
                  @within("execution(public String org.koushik.get() )")
                  …
         }
         @after("execution(public String org.koushik. synchronized block() )")
}
```

In above example, different advices are mentioned that have to execute at different points of synchronized block. *'get()'* is a method which we want to execute at different points of *'synchronized block()'*. Whenever *'synchronized block()'* executes, the advices run at certain points that are *'before()'*, *'within()'*, *'around()'*, and *'after()'*. By applying an advice on a defined points, the synchronized block executes deterministically.

'@before("execution(public String org.koushik.get())")' advice will run the *'get()'* just before execution of *'synchronized block()'*. *@within("execution(public String org.koushik.get())")* advice will run the *'get()'* within the body of *'synchronized block()'*. *@after("execution(public String org.koushik.get())")*

advice will run the *'get()'* just after completion of '*synchronized block()*'. By applying such advices of execution of different methods at different points in synchronized block we can get the visibility and ordering execution of threads.

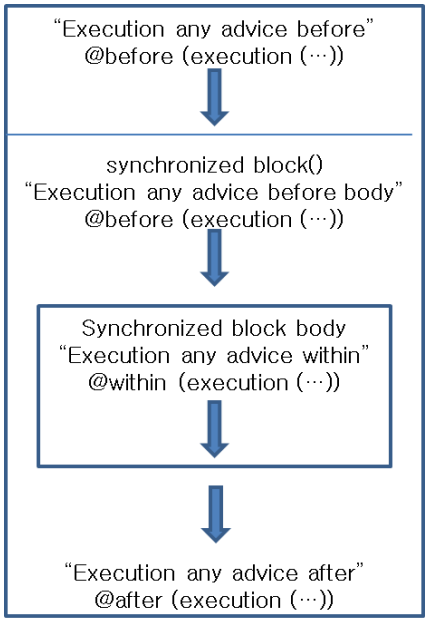

Figure 1. Flow of Aspect-Oriented Synchronization

Figure 1 shows the flow of execution of aspect advices. *'@before'* execute the defined process before synchronized block, after that process the synchronized block will execute. *'@within'* execute any given process within the body of block. *'@after'* execute any given process after the body of block. By using the advices we can obtain the deterministic execution of synchronized block.

6. CONCLUSION:

Synchronization holds high importance in a multithreading program because of sequential handling of threads. It is a big challenge to avoid the race-condition and deadlock from multithread system. To get synchronization in multithreaded program code, locking system has to be applied. However, there are high chances of deadlock and it also limits the scope of code. In this paper, we presented an AOP-based synchronization approach in concurrent systems with an example code. We demonstrated the use of constraints and advices made by using AOP, which can be applied on multiple processes. AOP-based synchronization can increase determinism in a multithreaded system with less chances of race-condition or deadlock occurrence.
In the future, we will do the comparison of AOP-based synchronization with lock-based approach over a real-time multithreaded program.

ACKNOWLEDGMENT

This work was supported by the ICT R&D program of MSIP/IITP. [R0601-15-1063, Software Platform for ICT Equipments]

REFRENCES

[1] Oßner, Christopher, and Klemens Böhm. "Graphs for Mining-Based Defect Localization in Multithreaded Programs." International Journal of Parallel Programming, pp.570-593, 2013
[2] Tanter, Éric, Ismael Figueroa, and Nicolas Tabareau. "Execution levels for aspect-oriented programming: Design, semantics, implementations and applications." Science of Computer Programming, pp.311-342, 2014
[3] Liu, Tongping, Charlie Curtsinger, and Emery D. Berger. "Dthreads: efficient deterministic multithreading". Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM, pp.327-336, 2011
[4] Xi, Chenchen, Bruno Harbulot, and John R. Gurd. "Aspect-oriented support for synchronization in parallel computing." Proceedings of the 1st workshop on Linking aspect technology and evolution. ACM, pp.1-5, 2009
[5] Marques, Eduardo RB, Francisco Martins, and Miguel Simões. "Cooperari: a tool for cooperative testing of multithreaded Java programs." Proceedings of the 2014 International Conference on Principles and Practices of Programming on the Java platform: Virtual machines, Languages, and Tools. ACM, pp.200-206, 2014
[6] Kienzle, Jörg, and Rachid Guerraoui. "Aop: Does it make sense? The case of concurrency and failures." European Conference on Object-Oriented Programming. Springer Berlin Heidelberg, pp.37-61, 2002
[7] Babamir, Seyed Morteza, Elmira Hassanzade, and Mona Azimpour. "Predicting potential deadlocks in multithreaded programs." Concurrency and Computation: Practice and experience, pp.5261-5287, 2015