Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 5.258

IJCSMC, Vol. 5, Issue. 4, April 2016, pg.436 – 443

DYNAMIC SUPPORT FOR DEADLOCK DETECTION

Gowdhami.D[1], Aruna Devi.P[2]

[1]Research Scholar, Dr SNS Rajalakshmi College of Arts and Science, Coimbatore

gowdhami.d@gmail.com [1]

[2]Assistant Professor, Computer Technology Department, Dr SNS Rajalakshmi College of Arts and Science, Coimbatore

aruna7825@gmail.com [2]

Abstract: Deadlock is one of the most serious problems in distributed systems environment and detection of deadlock has undergone extensive study. Now days many industries are developing an application program using database by sending Structured Query Language (SQL) statements to them and also finding data by executing the SQL Statements. If the applications are using the same database concurrently sometimes database deadlock will be occurred. Testing applications to determine how they cause database deadlocks is important as part of confirming correctness, reliability, and performance of these applications. But it is very difficult to reproduce database deadlocks.

Keywords: Deadlock, Software Testing

I. INTRODUCTION

Software Testing is the process of executing a program or system with the intent of finding errors or, it involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results Software is not unlike other physical processes where inputs are received and outputs are produced. Where software differs is in the manner in which it fails. Most physical systems fail in a fixed (and reasonably small) set of ways. By contrast, software can fail in many bizarre ways. Detecting all of the different failure modes for software is generally infeasible.

Unlike most physical systems, most of the defects in software are design errors, not manufacturing defects. Software does not suffer from corrosion, wear-and-tear -- generally it will not change until upgrades, or until obsolescence. So once the software is shipped, the design defects or bugs will be buried in and remain latent until activation.

Software bugs will almost always exist in any software module with moderate size: not because programmers are careless or irresponsible, but because the complexity of software is generally intractable and humans have only limited ability to manage complexity.

Testing should systematically uncover different classes of errors in a minimum amount of time and with a minimum amount of effort. A secondary benefit of testing is that it demonstrates that the software appears to be working as stated in the specifications. The data collected through testing can also provide an indication of the software's reliability and quality. But, testing cannot show the absence of defect -- it can only show that software defects are present.

A deadlock is defined as the situation where two or more processes permanently block each other by acquiring the lock on a resource that has already been requested by another process. This means one process is trying to lock the resource while the other one has already locked it.

The scope of research is to identify the deadlock in application. This system will identify the deadlock in the database and also in the multithreading application. This system will produce the result in chart and also in through the message. This will help when and where the deadlock was occurred in application.

II. OVERVIEW OF DATABASE DEADLOCK

In a multi-process system, deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.

For example, assume a set of transactions {T0, T1, T2, ...,Tn}. T0 needs a resource X to complete its task. Resource X is held by T1, and T1 is waiting for a resource Y, which is held by T2. T2 is waiting for resource Z, which is held by T0. Thus, all the processes wait for each other to release resources. In this situation, none of the processes can finish their task. This situation is known as a deadlock.

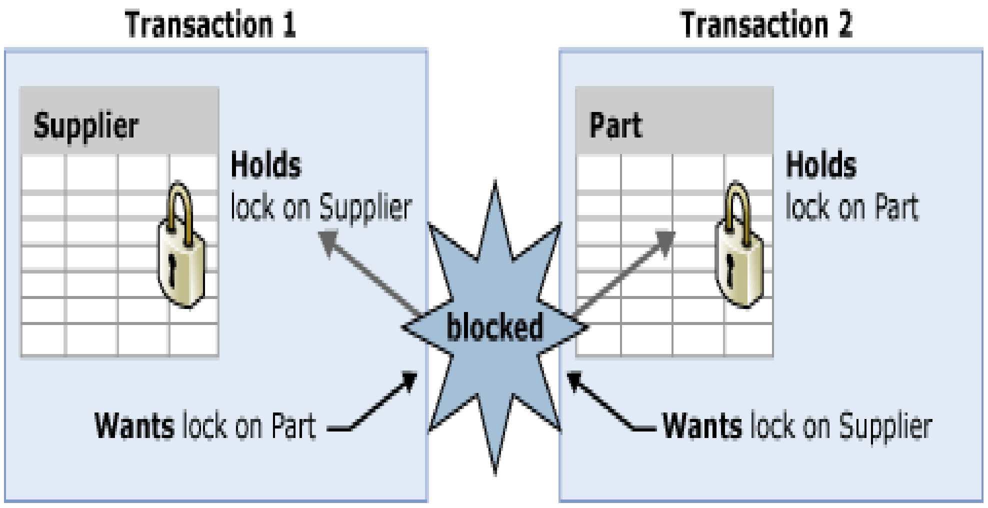
Deadlocks are not healthy for a system. In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

III. DEADLOCK IN SQL SERVER

Sometimes, a deadlock state is also called Cyclic Dependency as, process P1 depends on the process P2. In SQL Server, database engine automatically detects deadlock cycles. The SQL Server database engine chooses one of the sessions as the deadlock victim and the current transaction is terminated to break the deadlock.

Here, the process refers to a transaction. By default SQL Server transaction doesn't stop unless LOCK_TIMEOUT is set.

Deadlock can occur on any system having multiple threads not just on relational database management system. When we talk about database engine, sessions can encounter deadlock when acquiring non database resources like; memory or threads.



The above figure shows that Transaction 1 has acquired a lock on resource Supplier which Transaction 2 wants to acquire. However, Transaction 2 has acquired a lock on resource Part which Transaction 1 wants to acquire. Both are dependent on each other. There is a deadlock between T1 and T2.

IV. OUR APPROACH

The goal of proposed system is to detect the deadlock in application oriented project. Here we concentrate on both the Multithreading concept and also with the database connectivity. In our system we used Timestamp (i.e. Threshold Limit) for the executing the application and also we are showing result as chart based.

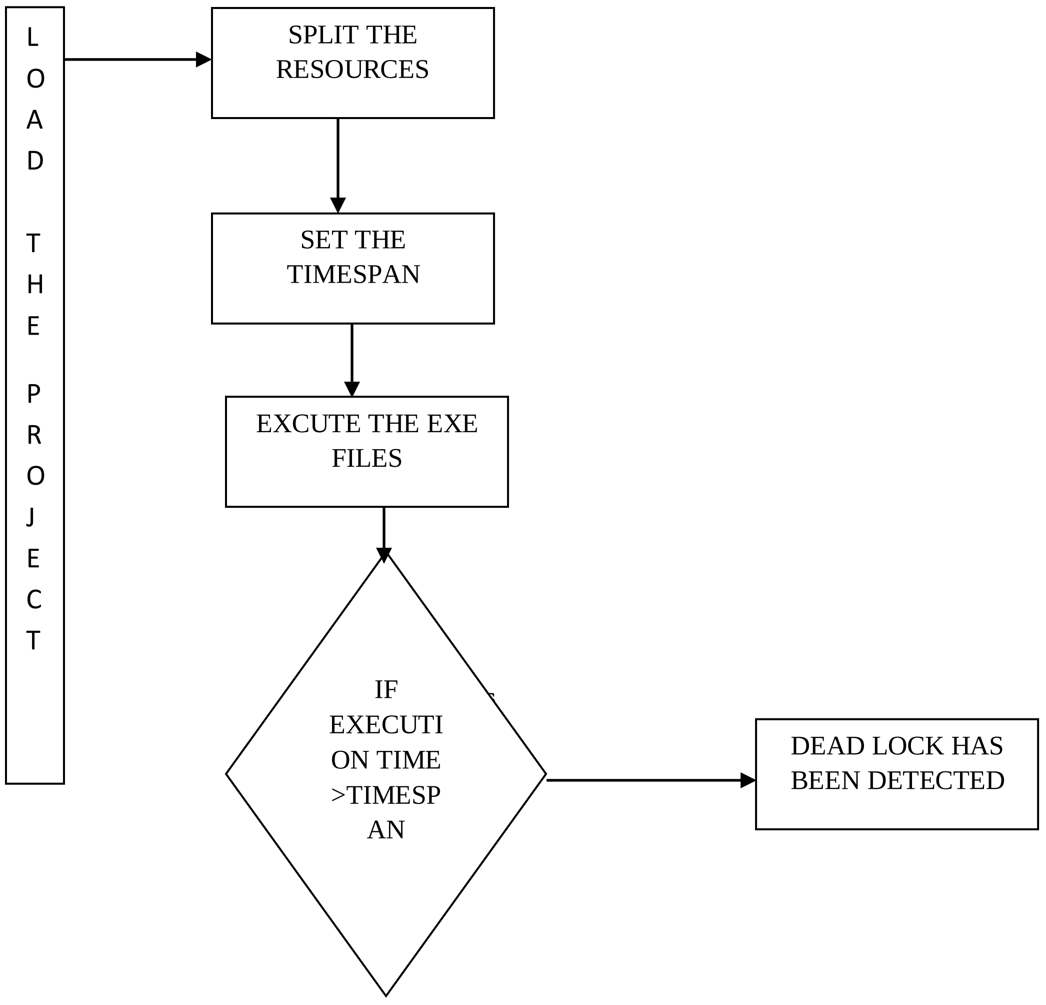
1. *Algorithm Used In Our System*

Here we used new algorithm named as Runway detector.

```
LOAD PROJECT SOURCE IN THE TOOL $\sum_1^n$ ps
VAR RUNAPP ←MAP ($\sum_1^n$ ps(RESFILE))
RUNAPP ←SELECT RESFILE(PARENTFILE)
EXECUTE SHELL(RUNAPP)
LOAD EXTERNAL LIBRARY AND OPEN RUNAPP
SET THE THRESHOLDLIMIT ← 1000 VALUE
IF EXECUTION TIME > THRESHOLDLIMIT  THEN
APPLICATION WILL BE QUIT AND DEADLOCK WILL OCCUR
```

```
ENDIF
IF DEADLOCK DETECT WITHIN THRESHOLDLIMIT THEN
PROPER MESSAGE WILL BE DISPLAYED WHERE THE DEADLOCK OCCURRED
END IF
```

2. Architecture of the System



V. IMPLEMENTATION AND RESULTS

1. Deadlock In SQL Server

Consider the example of database deadlock shown in Table4.1. Transactions T1 and T2 are independently sent by DCAs to the same database at the same time. When the first DCA executes the UPDATE statement in Step 1, the database locks rows of table employee in which the value of attribute employeeno is 1. Next, the second DCA executes the UPDATE statement in Step 2 and the database locks rows of table salary in which attribute employeeno is 2. When the SELECT statement in Step 3 is executed as part of transaction *T*1, the database attempts to obtain a read lock on the rows of table titles, which are exclusively locked by transaction *T*2 of the second DCA.

Since these locks cannot be imposed simultaneously on the same resource (i.e., these locks are not compatible), *T*1 is put on hold. Finally, the SELECT statement in Step 4 is executed as part of transaction *T*2; the database attempts to obtain a read lock on the rows of table authors, which are exclusively locked by transaction *T*1 of the first DCA. At this point both*T*1 and *T*2 are put on hold resulting in a database deadlock.

Step	**Transaction 1**	**Transaction 2**
1	UPDATE employee SET departmemt=HR WHERE employeeno=1	
2		UPDATE salary SET Basicpay=12000 WHERE employeeno=2
3	SELECT name, doj FROM Employee WHERE employeeno=2	
4		SELECT employeeno FROM salary WHERE basicpay >10000

Example for Deadlock

For SQL deadlock detection two real time applications were developed and tested using our proposed algorithm. Here two real time application has been tested for deadlock detection

A. Departmental Stores

In departmental store we used the client server approach, here the database is centric and application may be used in different system. This kind of deadlock will occur when two or three clients are generating the bill at the same time and update the record one after another. If the three clients generating the bill at same time with the same billing number. In this situation database deadlock will occur.

B. Invoice Billing

Here also the same client server approach has been used. In invoice billing the deadlock will occur when both product sales and purchase has been done at the same time, because the product table selection and updation done at simultaneously.

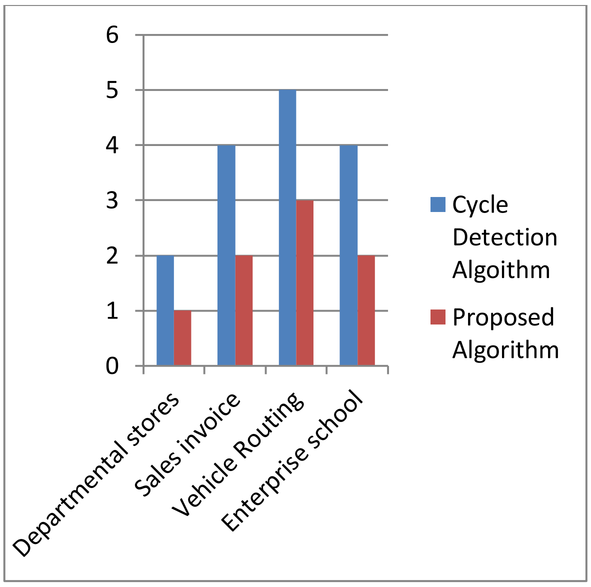
In database deadlock were not reproduced in regular baseline approach, but in our system it is possible to reproduce the deadlock.

2. Deadlock In Multithreading Application

For multithreading also two different applications has been tested for detecting the deadlock. They are vehicle Routing and Time Table Generator. 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 assignment. The deadlock monitor is control the overall process of program execution.

3. Result Analysis Comparison

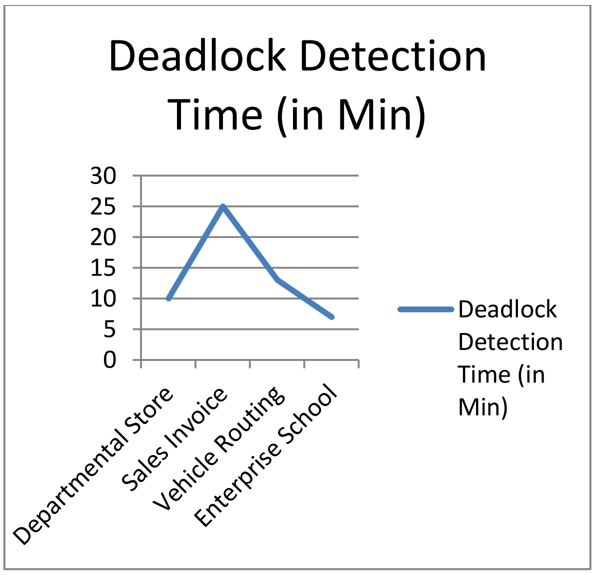
Below chart shows the difference between the two algorithms. It shows when the deadlock has been detected in those applications such as Departmental stores, Sales invoice, and Vehicle routing and enterprise school. Number (y-axis) says that number of execution. Testing has been done by using both the cycle detection algorithm and also the proposed algorithm



Comparison of Result with Existing System

4. Result Analysis Time Complexity

In the proposed tool we are using the time span for detecting the deadlock. By using the time span deadlock has been detected in those four applications such as departmental stores, sales invoice, vehicle routing and enterprise school,



Execution Time for Deadlock Detection

VI. CONCLUSION AND FUTURE SCOPE

Deadlock is a major problem in operating systems, Database and also concurrent programming. However there are several techniques to dead with deadlock such as deadlock avoidance, prevention etc. but still deadlock can occur. In this paper we have seen about the deadlock, database deadlock and how to detect deadlock detection and also various approaches used for deadlock detection and how to minimizing deadlock in database. And also we did testing in four different kinds of deadlock based real-time application.

It is not possible to avoid the deadlock in real world and also in real time application. So, in future we can do the testing with the same application in different approach using different algorithms to detect or identify the deadlocks in real time applications. And also we can do the volume testing and stress testing methodologies in future.

REFERENCES

[1] Indranil Roy. (2012). A scalable deadlock detection algorithm for UPC collective operations. *Iowa State University's High Performance Computing Group, Iowa State University, Ames, Iowa 50011, USA* . 2 (2), 145-167.

[2] Dr. P. Senthil Kumar. (2014). Indranil Roy. (2012). A scalable deadlock detection algorithm for UPC collective operations. Iowa State University's High Performance Computing Group, Iowa State University, Ames, Iowa 50011, USA .2 .*Professor, SKR Engineering College, Chennai – 600123, Tamil Nadu, India* . 3 (3), 730-733.

[3] T. Hilbrich. (2012). MPI Runtime Error Detection with MUST: Advances in Deadlock Detection. *Salt Lake City, UT, United States*. 2 (2), 296-305.

[4] Dorian C. Arnold.(2008). Stack Trace Analysis for Large Scale Debugging.*STAT's design and an evaluation*. 2 (1), 617-628.

[5] AnhVo .(2009). Scalable Verification of MPI Programs.*School of Computing, University of Utah, Salt Lake City, UT* . 22 (6), 56-66.

[6] R B SuriyaVaisshnavi. (2015). Dappled Deadlock Detection in Multithreaded Programs .*Deadlock Detection, Multithreaded Programs, Concurrency, Lock order graph, Scalability*. 2 (3), 520-534.

[7] Peng Liu. (2010). Context-Aware Fixing of Concurrency Bugs.*Context-aware fixing, concurrency bugs* . 1 (2), 139-154.

[8] Mahdi Eslamimehr. (2008). Sherlock: Scalable Deadlock Detection for Concurrent Programs. *UCLA, University of California, Los Angeles, USA*. 4 (1), 143-162.

[9] LengdongWu .(2014). Survey of Large-Scale Data Management Systems for Big Data Applications.*Department of Computing Science, University of Alberta, Edmonton, Alberta T6G2E8, Canada* . 5 (2), 761-772.

[10] IgnacioLaguna.(2006). ProbabilisticDiagnosisofPerformanceFaultsin Large-ScaleParallelApplications.*PurdueUniversity,SchoolofElectricalandComputerEngineering,WestLafayette,IN 47907*. 2 (7), 175-184.

[11] Duy-Khanh Le.(2011). An Expressive Framework for Verifying Deadlock Freedom.*Department of Computer Science, National University of Singapore*. 2 (1), 211-230.

[12] Sherif F. Fahmy.(2009). On Scalable Synchronization for Distributed Embedded Real- time Systems.*ECE Dept., Virginia Tech, Blacksburg, VA 24061, USA*. 5 (2), 259-268.

[13] Mark Grechanik. (2007). Testing Database-Centric Applications For Causes of Database Deadlocks. *University of Illinois at Chicago Chicago*. 1 (8), 154-168.

[14] Swati gupta(2014).Approaches for deadlock detection and prevention database distributed database system. *Amity university,India*.86-88

[15] SailenDutta Kalita1, Minakshi Kalita2,Sonia Sarmah3(2015),A Survey on Distributed Deadlock Detection Algorithm and its Performance Evolution.*Assam Don Bosco University, Azara,Guwahati,Assam*,India.616-620