Chapter 8.2 Deadlock

*Definition:*

*Deadlock is kind of Database Status which belongs to Currency Transactions because of resource competition: Each Transaction among several transactions is waiting for the resource which has been occupied by other Transactions, therefore each Transaction can not any progression.*

*Scenario:*

1. Even normal Operation of Two - Phase Locking (2PL) Transaction may cause Deadlock. The reason is that one Transaction has locked the resource which has been locked by another Transaction.
2. When the Lock has been updated from the Shared Lock into Exclusive Lock, then this may cause Deadlock. The reason is that, each transaction want to upgrade from the Shared Lock into Exclusive Lock of the Common Element.

*Method to Deal with Deadlock:*

*There have two methods to deal with Deadlock.*

* *The first one is to detect Deadlock and repair it.*
* *The other one is to manage Transaction, even Deadlock may not happen forever.*

Chapter 8.2.1 Timeout Deadlock Detection

*Principle:*

When exists Deadlock, then it is possible to repair all related Transactions and make them continue to execute. Since, one of the Transaction needs to abort and restart again.

*Method:*

*The simplest method to solve Deadlock is to utilize Timeout. We need to limit the time of Active Transaction, if the Transaction has exceeded this time, then this Transaction needs to be rolled back.*

*Supplement:*

Attention that, when one Deadlock Transaction has exceeded time and finished rolled back, then this Transaction needs to release its lock and all other Resources. Therefore, some other Transactions may finish before they reach the time limit.

Chapter 8.2.2 Waiting Image

*Principle:*

By using the Waiting Image to detect the Deadlock problem since one Transaction is waiting for the Lock occupied by another Transaction. The Waiting Image represents which Transaction is waiting for locks occupied by other Transactions. The Waiting Image can be used to detect Deadlock after it has been formed. Also, the Waiting Image can be used to prevent the formation of Deadlock. We assume that the latter one happens, in any time, we need to maintain the Waiting Image and decline the formation of Cycle.

*Definition:*

Waiting Image maintains Holding Lock and Waited Lock for each Transaction to be one Node. For each Node/Transaction T and U, if exists some Database Element which makes:

1. The Lock that Transaction U keeps on Database Element A.
2. The Lock that Transaction T waits for Database Element A.
3. Unless Transaction U release its Lock on A first, otherwise Transaction T can not get corresponding Lock.

As long as these three conditions have been satisfied, then there exists one arch from Transaction T to U.

* If there does not exist cycle in the Waiting Image, then each Transaction would finish at last. At least, one Transaction does not wait for another Transaction, then this Transaction can be finished.
* If there has cycle in the Waiting Image, then any Transactions in Cycle can not get progression, therefore exists the Deadlock.At that time, one Strategy to avoid Deadlock is to rolled back the Transaction which requirement may cause the cycle exists.

*Example:*

Assume that there exist four Transactions, each Transaction read one Element and write another Element:

T1: l1(A); r1(A); l1(B); w1(B); u1(A); u1(B);

T2: l2(C); r2(C); l2(A); w2(A); u2(C); u2(A);

T3: l3(B); r3(B); l3(C); w3(C); u3(B); u3(C);

T4: l4(D); r4(D); l4(A); w4(A); u4(D); u4(A);

We are using one simple Locking System. The table below is the most start part of the Schedule among four Transactions.

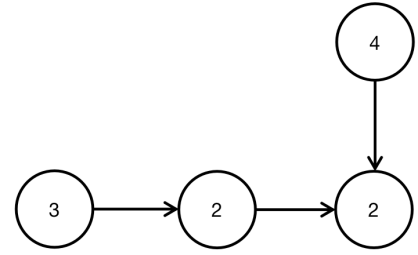
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Transaction T1 | Transaction T2 | Transaction T3 | Transaction T4 |
| 1 | l1(A); r1(A) |  |  |  |
| 2 |  | l2(C); r2(C); |  |  |
| 3 |  |  | l3(B); r3(B); |  |
| 4 |  |  |  | l4(D); r4(D); |
| 5 |  | l2(A) is declined. |  |  |
| 6 |  |  | l3(C) is declined; |  |
| 7 |  |  |  | l4(A) is declined; |
| 8 | l1(B); is declined; |  |  |  |

*Analysis:*

In the first four step, each Transaction gets the required Lock on which the Transaction wants to read.

In the fifth step, Transaction T2 tries to lock Database Element A, but since Transaction T1 has kept Lock on Database Element A, therefore the requirement has been declined. Therefore, Transaction T2 is waiting for Transaction T1. There would exists one Arch from Transaction T2 to Transaction T1.

In the sixth step, Transaction T3 tries to lock Database Element C, but since Transaction T3 has kept Lock on Database Element C, and the requirement has been declined. Similarly, in the seventh step, Transaction T4 required Lock which has been kept by Transaction T1. At that moment, the Waiting Image is shown as below, in this image there has no cycle.

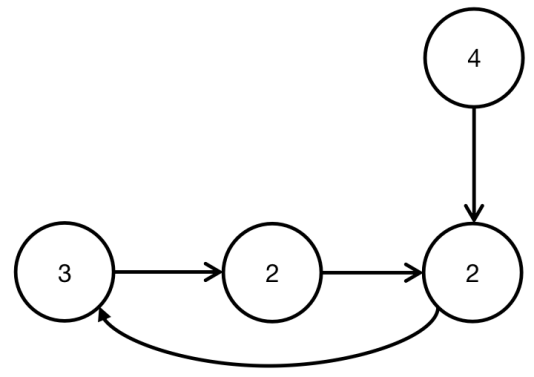


In the eighth step, here the Transaction T3 must be waiting for the lock on Database Parameter B. If we allow the Transaction T1 to wait, then there has one cycle among T1, T2 and T3, if the Transaction T1 is allowed to wait. Since each transaction is waiting for another transaction’s finishing, therefore, there exists the Deadlock among these three Transactions. Transaction T4 happens can not be finished, although it is not in the cycle, but it’s improvement still depends on the progress of Transaction T1.

Therefore, we need random Transaction to be rolled back which has caused cycle, therefore, Transaction T1 must be waiting. The generated Waiting Image is just like the picture above. Transaction T2 gives up lock on the Transaction T2 or T4.

Assume that lock can be given to Transaction T2, then Transaction T2 can finish. Then Transaction T2 can release the Lock on Database Element A and C. Transaction T3 needs Lock on Database Element C, and Transaction T2 needs the Lock on the Database Element A.

On some time, the Transaction T1 restarts, but before the finishing all Transaction T2, T3 and T4, it can not get the lock of Database Element A and B.



Chapter 8.2.3 Prevent Deadlock through Sorting Database Element

*Principle:*

Let’s consider several other methods to prevent Deadlock. The first method for us needs us to sort all Database Elements according to some kind of random or fixed Sequence. For example, If Database Element is block, then we can use their Physical Address to sort sequentially.

If Database Element which each Transaction is now applying for, then this would not cause Deadlock because there is still has Transaction waiting. To prove this, assume that Transaction T2 is waiting for the lock that Transaction T1 has on Database Element A1. And Transaction T3 is waiting for the Lock that Transaction T2 has.

*Example:*

Assume that all Database Elements are sorted alphabetically. If all following Transactions sort according to alphabetically, all Transaction T2 and T4 needs to be rewrite, then the sequence of Locked Transactions. These Transactions are now:

T1: l1(A) ; r1(A); l1(B); w1(B); u1(A); u1(B);

T2: l2(A); l2(C); r2(C); w2(A); u2(C); u2(A);

T3: l3(B); r3(B); l3(C); w3(C); u3(B); u3(C);

T4: l4(A); l4(D); r4(D); w4(A); u4(D); u4(A);

Transaction Sequence Table below gives same execution situation just as above. Transaction T1 gets lock on Database Element A. T2 continues to get lock on A, but it must wait for Transaction T1 to finish. Then, Transaction T3 gets lock on B, but Transaction T4 can not start, it also needs the lock on Database Element A while the lock has already been occupied by Transaction T1, so it must wait.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Transaction T1 | Transaction T2 | Transaction T3 | Transaction T4 |
| 1 | l1(A); r1(A); |  |  |  |
| 2 |  | l2(A); has been declined; | l3(B); r3(B); |  |
| 3 |  |  |  | l4(A); has been declined; |
| 4 |  |  | l3(C); w3(C); |  |
| 5 |  |  | u3(B); u3(C); |  |
| 6 |  |  |  |  |
| 7 | l1(B); w1(B); |  |  |  |
| 8 | u1(B); u1(B); |  |  |  |
| 9 |  | l2(A); l2(C); |  |  |
| 10 |  | r2(C); w2(A); |  |  |
| 11 |  | u2(A); u2(C); |  |  |
| 12 |  |  |  | l4(A); l4(D); |
| 13 |  |  |  | r4(D); w4(A); |
| 14 |  |  |  | u4(A); u4(D); |

Transaction T2 has been stopped, therefore it can not continue, but according to table above, the next one in the transaction Sequence is Transaction T3. Transaction T3 can get lock on Database Element.

Transaction T3 finishes in the sixth step. Now all locks on B and C of Transaction T3 have been released, Transaction T1 can finish in the eighth step.

At that time, Lock on A can be received, and we need to give lock to Transaction T2 according to first come first service. Transaction T2 gets two Locks it require and finishes in eleventh step. At last, Transaction T4 gets the lock and finishes execution.

Chapter 8.2.4 Deadlock Detection through Time Stamp

*Introduction:*

Just as described above, we can maintain Waiting Image to detect Deadlock. However, Waiting Image may be too large, and each time waiting for the transaction to analyze the Waiting Image and to check whether there is any cycle in the Waiting Image may cost a lot of time.

*Definition:*

There is another method that can be chosen which is to relate each Transaction with each Time Stamp. The Time Stamp can only be used to Detect Deadlock. Especially, if the Transaction rolled back, then it will restart with a new, later Concurrency Time Stamp, but Time Stamp which is used to detect Deadlock has never changed.

*Usage:*

When one Transaction T is waiting for the Lock which has been kept by another Transaction U, the Time Stamp must be used. According to whether the Transaction T or Transaction U is older or not, there may be two situations. These two strategies can be used to manage Transaction and detect Deadlock.

1. Wait - Dead Schema:
   1. Transaction T is older than Transaction U, then Transaction T can wait for Lock which is kept by Transaction U.
   2. Transaction U is older than Transaction T, then Transaction T need to die, and Transaction T go to rollback.
2. Hurt - Wait Schema:
   1. Transaction T is older than Transaction U, then Transaction T can kill Transaction U. Then Transaction U must rollback and give up all locks that belongs to Transaction T. Of course, there exists one exception which means that before the kill takes effect, then Transaction U has finished and released all its locks. Under this kind of situation, Transaction U can survive and does not need to rollback.
   2. Transaction T is older than Transaction U, then Transaction T is waiting for Transaction U to finish and then get all locks that Transaction T acquired.

*Example:*

Take the example into consideration. Let’s consider the Wait - Dead Schema. We assume that Transaction T1, T2, T3, and T4 are sorted based on the time, which means that Transaction T1 is the oldest Transaction. Also, in this example, we assume that when the Transaction rollback, the restart of this Transaction would not be so fast, it would not become active before other Transactions finish. Just as the table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | T1 | T2 | T3 | T4 |
| 1 | l1(A); r1(A); |  |  |  |
| 2 |  | l2(A); die. |  |  |
| 3 |  |  | l3(B); r3(B); |  |
| 4 |  |  |  | l4(A); die; |
| 5 |  |  | l3(C); w3(C); |  |
| 6 |  |  | u3(B); u3(C); |  |
| 7 | l1(B); w1(B); |  |  |  |
| 8 | u1(A); u1(B); |  |  |  |
| 9 |  |  |  | l4(A); l4(D); |
| 10 |  | l2(A); waits |  |  |
| 11 |  |  |  | r4(D); l4(D); |
| 12 |  |  |  | u4(A); u4(D); |
| 13 |  | l2(A); l2(C); |  |  |
| 14 |  | r2(C); w2(C); |  |  |
| 15 |  | u2(A); u2(C); |  |  |

*Analysis:*

Transaction T1 gets Lock on Database Element A. When Transaction T2 tries to get Lock on Database Element A, Transaction T2 dies, since Transaction T1 is much older than T2. In the third step, Transaction T3 gets Lock on Database Element B, but in the forth step, Transaction T4 asks Lock on Database Element A, it dies since Transaction T1 is much older than Transaction T4. Next, Transaction T3 gets Lock on Database Element C and finishes. When Transaction T1 continues, and it finds that the Lock on Database B can be achieved, then it finishes in the eighth step.

At that time, the Lock on A can be achieved. Therefore, Transaction T2 and T4 can restart. For Deadlock, their time stamp would not change. Transaction T2 is much older than Transaction T4. But, we assume that restart the Transaction T4 first, therefore, in the tenth step, Transaction T2 requires the lock on Database Element A, but the Lock has already granted on Database Element A, then Transaction T2 needs to wait. Transaction T4 finishes in the twelveth step, then Transaction T2 continues to the end, just as the process in the last three steps.

*Example:*

Next, we consider to run the same Transaction under Hurt - Wait Strategy, just as table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | T1 | T2 | T3 | T4 |
| 1 | l1(A); r1(A); |  |  |  |
| 2 |  | l2(A); waits |  |  |
| 3 |  |  | l3(B); r3(B); hurts |  |
| 4 |  |  |  | l4(A); waits |
| 5 | l1(B); w1(B); |  |  |  |
| 6 | u1(A); u1(B); |  |  |  |
| 7 |  | l2(A); l2(C); |  |  |
| 8 |  | r2(C); w2(A); |  |  |
| 9 |  | u2(A); u2(C); |  |  |
| 10 |  |  |  | l4(A); l4(D); |
| 11 |  |  |  | r4(D); w4(D); |
| 12 |  |  |  | u4(A); u4(D); |
| 13 |  |  | l3(B); r3(B); |  |
| 14 |  |  | l3(C); w3(C); |  |
| 15 |  |  | u3(B); u3(C); |  |

Transaction T1 locks Database Element A. In the second step, Transaction T2 required the lock on A, however, it needs to wait, since Transaction T1 is older than Transaction T2. In the third step, Transaction T3 gets the Lock on Database Element B, Transaction T4 needs to wait Lock on A.

Next, assume that Transaction T1 continues to execute and request Lock on Database Element B. However, the Lock has been kept by Transaction T3, but T1 is much older than T3. So it can hurts T3. Since Transaction T3 has not been finished, then this hurt can be deadly: Transaction T3 needs to give up its own lock and roll back. Therefore, Transaction T1 can finish.

After Transaction T1 releases Lock on Database Element A, then the lock can be achieved by Transaction T2, then Transaction T2 can continue to execute. After Transaction T2, the Lock can be achieved by Transaction T4, Transaction T4 can continue to execute to the end. At last, Transaction T3 restarts and finishes.

Chapter 8.2.5 Comparison of Deadlock Management Mechanism

In Wait - Dead and Hurt - Wait Schema, old Transaction kills newer Transaction. Since the Transaction restarts at Old Time Stamp, at the end, each Transaction would become the oldest transaction and finish. If the Transaction can finish at the end, then we call this kind of guarantee ‘No Starvation’. Attention that, other method can not prevent ‘*Starvation’*; But if not take extra method, then the Transaction may restart, deadlock and then rollback.

*There have tiny differences between two Schema, Wait - Dead and Hurt - Wait Schema:*

* *In Hurt - Wait Schema, as long as older Transaction requests Lock which the newer Transaction kept, then the newer Transaction will be killed. We can see that rollback is seldom in Hurt - Wait Schema.*
* *The Hurt - Wait Schema need to rollback much more than Wait - Dead Schema, but this Transaction has already occupied much more processor time. Therefore, according to the different number of Transactions, two Schema may waste much more work than the other one.*

*Advantage and Disadvantage of these two Schema. There have several points:*

* *The Realization of Wait - Dead and Hurt - Wait is much easier than maintain or construct Waiting Image.*
* *Waiting Image can minimize the times of Transaction Abortion because of Deadlock. If we abort Transaction, then there must has existed Deadlock. On the other way around, Wait - Dead and Hurt - Wait Schema may rollback Transaction although there doesn’t exist Deadlock.*