Distributed Database System (CS-600)

Mr. Nauman Iqbal email id: nauman@biit.edu.pk Whatsapp# 0321-5821151
Mr. Ahsan email id: ahsan@biit.edu.pk, Whatsapp# 0333-5189656

(Week 12) Lecture 23-24 (Chapter 20 of Book)

Assignemnt#10 (given at End)

1. **Submission Date: Sunday 24-May-2020 before 11:59PM.**
2. **You must prepare handwritten Assignment and send it to respective course teacher (after scanning it) for assessment by email only.**

Assignment submission through Email:

Email Subject must be in Correct format otherwise assignment will not be checked.

Email Subject Format: 4digitReg-DDS-section-ASG10

Example: 1234-DDS-CS6A-ASG10 Correct

2017-ARID-1234-DDS-CS6A-ASG10 Incorrect

Objectives: Learning objectives of this lecture are

- **Deadlock Detection Using Wait-For Graph**
- **Recovery from Deadlock**
- **Database Recovery after Failure event**

Distributed Database System (CS-600)

Mr. Nauman Iqbal **email id: nauman@biit.edu.pk Whatsapp# 0321-5821151**
Mr. Ahsan **email id: ahsan@biit.edu.pk, Whatsapp# 0333-5189656**

Overview:

In this lecture we will study how to detect a deadlock in a schedule? What are the factor to consider when system is stuck in a deadlock state? As transaction has to be killed to get rid of a deadlock, we will study database recovery as well. What are the factors of failure in database systems?

Recap:

In previous lecture, we have studied about deadlock. What is a deadlock in a schedule? "An impasse that may result when two (or more) transactions are each waiting for locks to be released that are held by the other". Deadlock creates an unlimited wait in the system due to which system get stuck.

There are certain techniques to prevent deadlock which we have studied. For example; time-out, wait-die, wound-wait and conservative two phase locking which we have studied in previous lecture.

Deadlock Detection:

- Transactions can enter a system anytime.
- A schedule consisting of number of transaction is bundled up with transaction to be executed by system.
- If a schedule contains a deadlock and it is in execution in the system, this may cause more damage in the system.
- A better solution is to check if schedule is executed as it is, whether deadlock will occur or not without executing the transactions.
- Deadlock detection is usually handled by the construction of a **wait-for graph** (WFG).
- It shows the transaction dependencies;
    - Transaction T*i* is dependent on T*j* if transaction T*j* holds the lock on a data item that T*i* is waiting for.

Wait-for Graph:

- The WFG is a directed graph.
- WFG consists of a set of nodes N (which represent transactions)
- A set of directed edges E (between nodes).

Wait-for Graph Creation Rules:

a. Create a node for each transaction.

b. Draw a directed edge from Transaction Ti → Tj;

Mr. Nauman Iqbal **email id: nauman@biit.edu.pk Whatsapp# 0321-5821151**
Mr. Ahsan **email id: ahsan@biit.edu.pk, Whatsapp# 0333-5189656**

- if transaction T*i* is waiting to lock an item that is currently locked by T*j*.

✓ If Wait-for Graph contains any cycle in it, either between two transactions or between more than two transactions, then there will be a **DEADLOCK** in a schedule.
✓ If there is no cycle in Wait-for Graph, then there is **NO-DEADLOCK** in schedule.

Example 1:

- Consider the given schedule having two transaction T1 and T2.
- Let us draw Wait-for Graph for deadlock detection step by step.

Time	T1	T2
t0	**Begin-transaction**	
t1	Lock(balx)	
t2	Read(balx)	**Begin-Transaction**
t3	balx=balx+100	Lock(baly)
t4	Write(balx)	Read(baly)
t5	Lock(baly)	baly=baly+50
t6	baly=baly*3	Write(baly)
t7	Write(baly)	Lock(balx)
t8	Commit	balx=balx / 2
t9		Write(balx)
t10		Commit

Step 1: Create Node for Each transaction

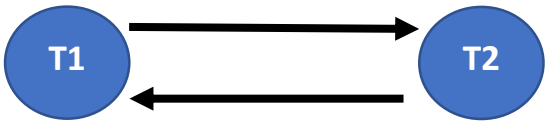
Step 2[Checking Operations of Transaction T1]:

- Draw a directed edge from transaction Ti → Tj, if Ti is waiting for a lock held by Tj.
- As we can see Transaction T1 is locking (balx) at time t0, as balx is not held by any other transaction so T1 will not wait here.
- Afterwards, T1 is requesting for Lock(baly) at time t5. but as we can see, baly is already locked by transaction T2 at time t3.so T1 has to **WAIT.**
- As T1 has to wait for T2, so a directed arrow will be drawn from T1 → T2.

Mr. Nauman Iqbal **email id: nauman@biit.edu.pk Whatsapp# 0321-5821151**
Mr. Ahsan **email id: ahsan@biit.edu.pk, Whatsapp# 0333-5189656**

Step 3[Checking Operations of Transaction T2]:

- Draw a directed edge from transaction Ti → Tj, if Ti is waiting for a lock held by Tj.
- As we can see, Transaction T2 is locking (baly) at time t3. At that time baly is not allocated to any other transaction so T2 will not wait.
- Afterwards, Transaction T2 is locking (balx) at time t6. At that time, balx is already assigned to transaction T1.
- As T2 has to wait for transaction T1 for balx, so a directed arrow will be drawn from T2→T1.



Step 4 [Operations of All Transaction has been checked]

- Now we will check if graph has cycle in it or not.
- As we can see, we can move from T1 to T2 using upper arrow.
- And back to T1 using lower arrow.
- Which means there is a cycle in the graph.

Step 5 [Declaration]

- As there is cycle between T1 and T2 so Schedule has **"Deadlock".**

Example 2:

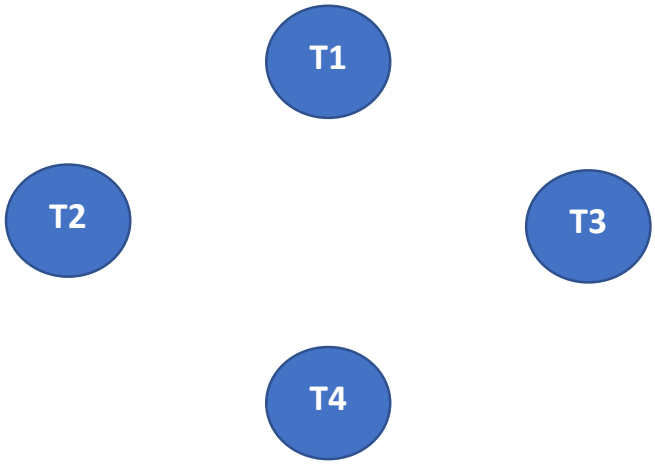
- Consider the given schedule having two transaction T1, T2, T3 and T4.
- Let us draw Wait-for Graph for deadlock detection step by step.

Time	T1	T2	T3	T4
t_0			**Begin Tran**	**Begin Tran**
t_{01}	**Begin Tran**	**Begin Tran**	Lock(Y)	
t_{02}	Lock(X)	Lock(T)	Y=Y*2	
t_{03}	X=X+10	T=T-20	Y=Y+100	Lock(S)
t_{04}	Write(X)	Lock(B)	Lock(A)	Read(S)
t_{05}	Lock(A)	Read(B)	Read(A)	S=S+150
t_{06}	Read(A)	Lock(E)	Lock(B)	Lock(T)
t_{07}	A=A+50	Read(E)	Read(B)	Read(T)
t_{08}	Commit	E=E+80	B++	T=T+S
t_{09}		Commit	Write(B)	Commit
t_{10}			Commit	

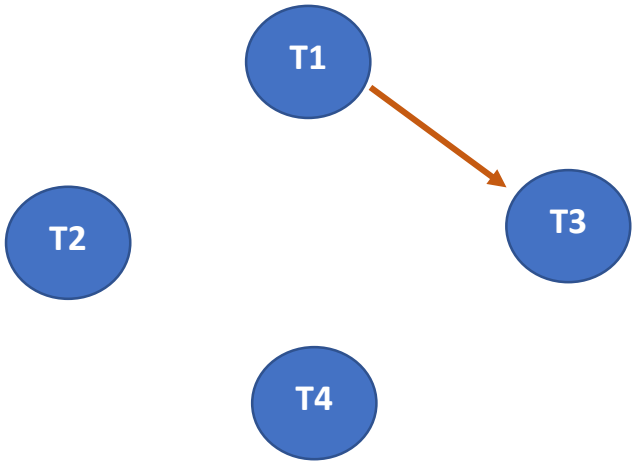
Distributed Database System (CS-600)

Mr. Nauman Iqbal email id: nauman@biit.edu.pk Whatsapp# 0321-5821151
Mr. Ahsan email id: ahsan@biit.edu.pk, Whatsapp# 0333-5189656

Step 1: Create Node for Each transaction

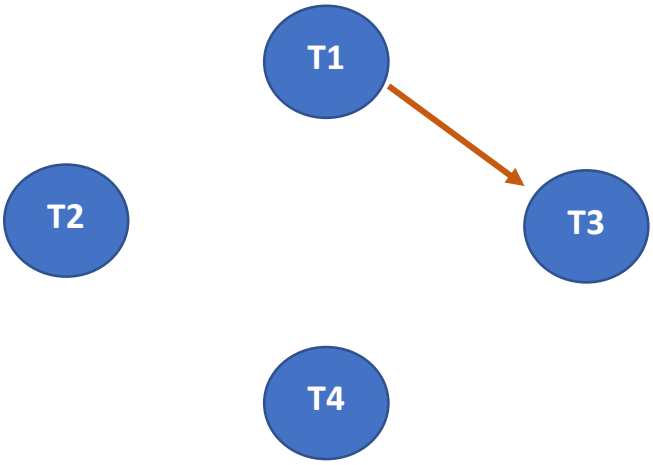


Step 2: Checking Transaction T1 for Waiting operations



- T1 waiting for transaction T3 for data-item "a".
- Because T3 has allocated "a" before T1.
- So a directed edge from T1 → T3

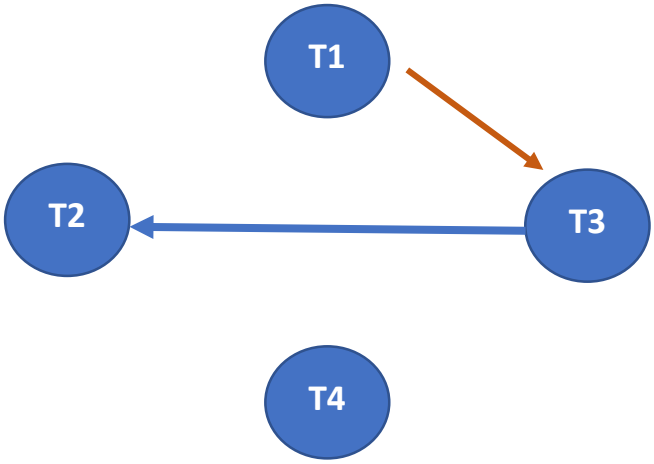
Step 3: Checking Transaction T2 for Waiting operations



- T2 will not wait for any transaction because it has been allocated all locks it required.

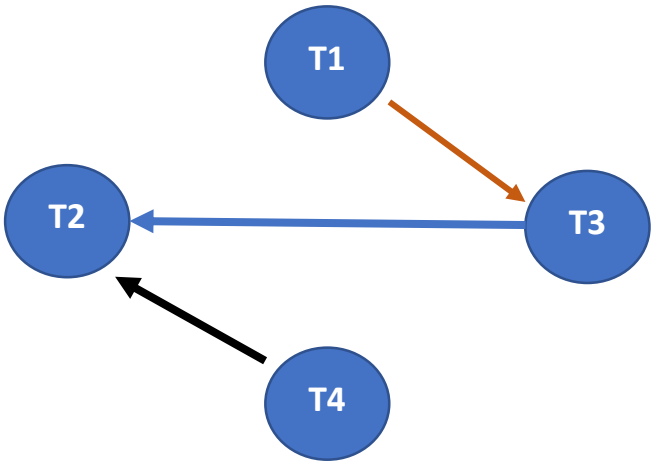
Mr. Nauman Iqbal **email id: nauman@biit.edu.pk Whatsapp# 0321-5821151**
Mr. Ahsan **email id: ahsan@biit.edu.pk, Whatsapp# 0333-5189656**

Step 4: Checking Transaction T3 for Waiting operations



- T3 will wait for transaction T2 for data-item 't'.
- Because T2 has locked "t" before T3. So T3 will have to wait
- So a directed edge from T3→T2.

Step 5: Checking Transaction T4 for Waiting operations



- T4 will wait for Transaction T2 for data-item "t".
- Because T2 has allocated "t" before transaction T4.
- So a directed edge from T4 → T2

Step 5 [Declaration]

- As we can see, there is no cycle in above graph.
- So there is **"No-Deadlock"** in given schedule.

Recovery from deadlock detection:

- Once deadlock has occurred in system, there is only one solution to get rid of deadlock that is to kill(rollback) any one of the transaction.
- As we have studied in previous lecture about techniques that prevent deadlock.
- But each technique has some drawbacks which may cause irrelevant transaction to rollback.

Mr. Nauman Iqbal **email id: nauman@biit.edu.pk Whatsapp# 0321-5821151**
Mr. Ahsan **email id: ahsan@biit.edu.pk, Whatsapp# 0333-5189656**

- Those techniques can also cause more damage in system by not selecting the correct victim.
- Keeping in mind above facts, there are some factors which needs to be taken care off while getting rid of deadlock in any system.

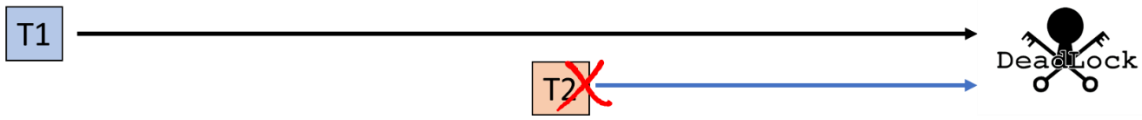
1- **Choice of Deadlock Victim**
2- **How far to rollback transaction back**
3- **Avoid Starvation**

- Now let us discuss above three factors one by one.

1- Choice of Deadlock Victim

In some circumstances, the choice of transactions to abort may be obvious. However, in other situations, the choice may not be so clear. In such cases, we would want to abort the transactions that incur the minimum costs. This may take into consideration:

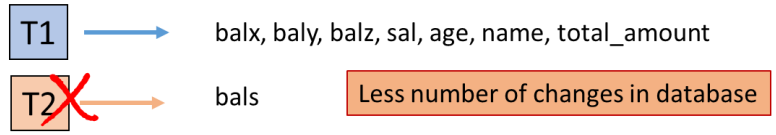
- **How long the transaction been running?**
  - Simply priority must be given to the transaction that has been in execution for longer period of time.
  - Those transactions which are younger would be more obvious to be killed by system.



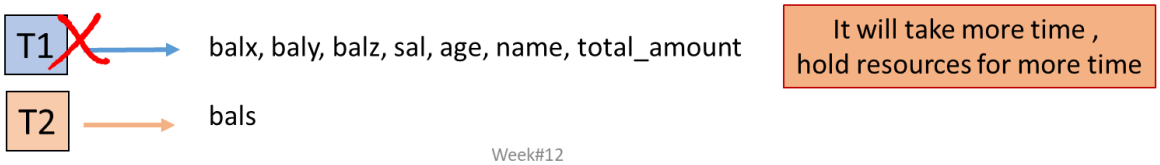
- Here we can see that T2 is executed lesser than T1 so T2 has to be killed.

- **How many data-items have been updated**
  - Those transactions which have updated more data-items will be given priority over those who have updated less data-items.
  - The reason behind is if transaction which has updated more data-items will be killed, then DBMS has to make more number of changes in database.
  - On the other hand, we will prefer less number of changes in case of rollback so that system load must be reduced.

Mr. Nauman Iqbal **email id: nauman@biit.edu.pk Whatsapp# 0321-5821151**
Mr. Ahsan **email id: ahsan@biit.edu.pk, Whatsapp# 0333-5189656**



- In above figure we can see that T1 has updated 7 data-items, if we rollback it system will have to change 7 data-items in database.
- On the other-hand T2 has only updated one data-item (bals), so if it is rollback then only one item has to be changed.
- **How many data items the transaction is still to update?**
  - It would be better to abort a transaction that has many changes still to make to the database rather than one that has few changes to make.
  - Unfortunately, this may not be something that the DBMS would necessarily know.



  - In above figure, T1 has still to update 7 data-items which will take a long time.
  - On the other-hand, T2 only has one data-item to update so it will take less time.
  - This is why T1 should be killed.

Victim will be chosen by considering above three points. It is always favorable to make less changes in database in case of rollback.

2- How far to rollback transaction back

- having decided to abort a particular transaction, we have to decide how far to roll the transaction back.
- Clearly, undoing all the changes made by a transaction is the simplest solution, although not necessarily the most efficient.
- It may be possible to resolve the deadlock by rolling back only part of the transaction.

3- Avoid Starvation

- when the same transaction is always chosen as the victim, and the transaction can never complete.
- Starvation is basically selection of same transaction for a deadlock victim again and again. So that transaction never gets a chance to be completed.

Mr. Nauman Iqbal email id: nauman@biit.edu.pk Whatsapp# 0321-5821151
Mr. Ahsan email id: ahsan@biit.edu.pk, Whatsapp# 0333-5189656

- The DBMS can avoid starvation by storing a count of the number of times a transaction has been selected as the victim and using a different selection criterion once this count reaches some upper limit.

Timestamping:

- An algorithm which avoids starvation.
- Each transaction is assigned a Timestamp at which it appears.
- If a transaction is killed in a conflict of deadlock. It must re-appear in system to get executed.
- When transaction re-appear in system for execution, it is not assigned new timestamp but its actual timestamp when it arrived first in system.
- By doing so, transaction will get older by the passage of time. This will increase priority of transaction so that it can be given chance to get executed rather than getting killed in case of deadlock.

Database Recovery

Database recovery: The process of restoring the database to a correct state in the event of a failure.

- In previous lectures we introduced the concept of database recovery as a service that should be provided by the DBMS to ensure that the database is reliable and remains in a consistent state in the presence of failures.
- In this context, reliability refers to both the resilience of the DBMS to various types of failure and its capability to recover from them.
- In this section we consider how this service can be provided.
- To gain a better understanding of the potential problems we may encounter in providing a reliable system, we start by examining the need for recovery and the types of failure that can occur in a database environment.

The Need for Recovery

- The storage of data generally includes four different types of media with an increasing degree of reliability:

  i. main memory

  - Main memory is volatile storage that usually does not survive system crashes.

Mr. Nauman Iqbal email id: nauman@biit.edu.pk Whatsapp# 0321-5821151
Mr. Ahsan email id: ahsan@biit.edu.pk, Whatsapp# 0333-5189656

---

ii. magnetic disk

- Magnetic disks provide online non-volatile storage. Compared with main memory, disks are more reliable and much cheaper, but slower by three to four orders of magnitude.

iii. magnetic tape

- Magnetic tape is an offline non-volatile storage medium, which is far more reliable than disk and fairly inexpensive, but slower, providing only sequential access.

iv. optical disk.

- Optical disk is more reliable than tape, generally cheaper, faster, and providing random access.

- Main memory is also referred to as primary storage and disks and tape as secondary storage.
- Stable storage represents information that has been replicated in several non-volatile storage media (usually disk) with independent failure modes.
- For example, it may be possible to simulate stable storage using RAID (Redundant Array of Independent Disks) technology, which guarantees that the failure of a single disk, even during data transfer, does not result in loss of data.
- There are many different types of failure that can affect database processing, each of which has to be dealt with in a different manner.
- Some failures affect main memory only, while others involve non-volatile (secondary) storage.

Among the causes of failure are:

- **System crashes**
  due to hardware or software errors, resulting in loss of main memory.
- **Media failures**
  such as head crashes or unreadable media, resulting in the loss of parts of secondary storage.
- **Application software errors**
  such as logical errors in the program that is accessing the database, which cause one or more transactions to fail.
- **Natural physical disasters**
  such as fires, floods, earthquakes, or power failures.
- **Carelessness**
  unintentional destruction of data or facilities by operators or users.
- **Sabotage**
  or intentional corruption or destruction of data, hardware, or software facilities.

Mr. Nauman Iqbal **email id: nauman@biit.edu.pk Whatsapp# 0321-5821151**
Mr. Ahsan **email id: ahsan@biit.edu.pk, Whatsapp# 0333-5189656**

- Whatever the cause of the failure, there are two principal effects that we need to consider the loss of **main memory**, including the database **buffers**, and the loss of the **disk copy of the database**.
- In the next lecture we will discuss the concepts and techniques that can minimize these effects and allow recovery from failure.

Assignment # 10

Question No 1:

a- Consider the given schedule and tell if **deadlock** exists or not by creating **"Wait for Graph"**.

Mr. Nauman Iqbal **email id: nauman@biit.edu.pk Whatsapp# 0321-5821151**
Mr. Ahsan **email id: ahsan@biit.edu.pk, Whatsapp# 0333-5189656**

---

b- Re-arrange operations of the transaction given in schedule in such a way that deadlock must not occur in the given schedule. **You are only allowed to change sequence of operations in a transaction.** You are not allowed to delete or insert new operations. Rewrite the new schedule again and construct its "Wait-for Graph". New graph must not have any deadlock in it.

time	T1	T2	T3	T4	T5	T6
t_0			**Begin Tran**			**Begin Tran**
t_{01}		**Begin Tran**	Lock(Y)		**Begin Tran**	Lock(E)
t_{02}		Lock(T)	Read(Y)	**Begin Tran**	Lock(Z)	Read(E)
t_{03}	**Begin Tran**	Read(T)	Y=Y*2	Lock(S)	Read(Z)	E=E+44
t_{04}	Lock(X)	T=T-20	Y=Y+100	Read(S)	Z=Z+50	Lock(A)
t_{05}	Read(X)	Lock(B)	Lock(C)	S=S+150	Write(Z)	Read(A)
t_{06}	X=X+10	Read(B)	Read(C)	Lock(T)	Lock(Y)	A=E+50
t_{07}	Write(X)	Lock(E)	Lock(Z)	Read(T)	Read(Y)	Commit
t_{08}	Lock(A)	Read(E)	Read(Z)	T=T+S	Y=Y*5	
t_{09}	Read(A)	E=E+80	Z++	Commit	Lock(T)	
t_{10}	A=A+50	Commit	Write(Z)		Read(T)	
	Commit		Commit		Commit	

Question No 2: Solve the Lab Manuals attached with the lecture of Week 12.