Muhammd Rehan | 22P-9106 | BSE – 6A
Advanced Database Concepts
HomeWork 01

Timestamp Based Concurrency Control:- Wait Die and Wound Wait Protocols

Intro:

Deadlocks commonly occur in systems with database management under high transaction throughput. A deadlock arises when transactions are in a way that they are competing for resources held by one another, creating a condition that may cause execution to hang indefinitely. These deadlocks are prevented by integrating a simple systematic approach called the timestamp-based concurrency control, whereby upon starting a transaction, each transaction is assigned another timestamp- a unique identifier. Among a whole bunch of schemes of deadlock prevention, two basic ones: Wait-Die and Wound-Wait play central roles in regulating third party access to resources, based on juxtaposing the comparative ages of transactions as stipulated by their timestamps.

Timestamp Assignment in DBMS:

On starting a transaction, the system assigns that transaction a timestamp, which acts as a unique identifier. The uniqueness of a timestamp signifies the transaction's time of start, thereby enforcing a strict total ordering between transactions. Older transactions are those with lower timestamps, and younger transactions have higher timestamps.

Deadlock Prevention Strategies

In concurrent database access environments, transaction would often contend for the same resources. Given the weak control strategy, circular waiting can cause a deadlock. The Wait-Die and Wound-Wait protocols prevent this by adopting a timestamp-based priority resolution scheme

---

Wait-Die Protocol

The Wait-Die protocol is, in itself, a very simple non-preemptive method for deadlock handling.

Mechanism

1. If an older transaction (Ti) requests a resource held by a younger transaction (Tj), it is allowed to wait.
2. If a younger transaction (Tj) requests a resource held by an older transaction (Ti), Tj is aborted (dies) and restarted with the same timestamp.
3. The younger transaction will continue restarting until the conflict is resolved.

Key Principle:

Older transactions wait; younger transactions die and restart.

Example: Fund Transfers

Consider a banking system where two transactions, T1 (TS=5) and T2 (TS=10), attempt concurrent fund transfers involving Account A and Account B.

- Case 1: Older transaction requests a resource held by a younger transaction
  - T1 (older) requests Account A, which is locked by T2 (younger).
  - Since T1 is older, it is allowed to wait.
- Case 2: Younger transaction requests a resource held by an older transaction
  - T2 (younger) requests Account B, which is locked by T1 (older).
  - T2 is aborted and restarted.

Wound-Wait Protocol

The Wound-Wait protocol is a preemptive method of handling deadlocks.

Mechanism:

1. If an older transaction (Ti) requests a resource held by a younger transaction (Tj), Tj is aborted (wounded), and Ti preempts (takes) the resource.
2. If a younger transaction (Tj) requests a resource held by an older transaction (Ti), it is allowed to wait.

Key Principle:

Older transactions wound younger transactions by aborting them, while younger transactions wait for older ones.

Example: Fund Transfers

Using the same banking scenario, we observe the following:

- Case 1: Older transaction requests a resource held by a younger transaction
  - T1 (older) requests Account A, which is locked by T2 (younger).
  - T2 is aborted (wounded), and T1 acquires the resource immediately.
- Case 2: Younger transaction requests a resource held by an older transaction
  - T2 (younger) requests Account B, which is locked by T1 (older).
  - T2 waits until T1 releases the resource.

Feature	Wait Die	Wound Wait
Approach	Non-preemptive	Preemptive
Older Transaction Requests Resource	Waits	Aborts younger transaction
Younger Transaction Requests Resource	Aborts (Dies)	Waits
Deadlock Prevention	Yes	Yes
Overhead	More restarts	More preemptions

Efficiency Considerations

While both protocols effectively prevent deadlocks, their performance implications differ:

- Wait-Die leads to frequent transaction restarts, increasing computational overhead.
- Wound-Wait results in fewer restarts but may cause higher rollback overhead due to forced preemptions.
- Wound-Wait is often preferred in high-throughput environments due to its reduced waiting time for older transactions.

Practical Applications

These concurrency control techniques are widely implemented in:

- Database Management Systems (DBMS): PostgreSQL, MySQL, and Oracle DB incorporate variations of timestamp-based scheduling.
- Distributed Computing: Ensures transactional ordering in blockchain and distributed ledgers.
- Operating Systems: Facilitates resource allocation in multi-threaded environments.