Databases

Laboratorio de sistemas distribuidos

Universidad Politécnica de Madrid (UPM)
http://lsd.ls.fi.upm.es/lsd/lsd.htm
Summary

• Transactions.
• Isolation. Concurrency control.
• Isolation levels.
• Database replication.

Nuevas tendencias en sistemas distribuidos
Motivation

• What happens if the two programs are executed concurrently?

1
a=Read(A)
if a >= 100 then
Write(A,a-100)
b=Read(B)
Write(B,b+100)
end if

2
a=Read(A)
if a >= 100 then
Write(A,a-100)

3
4
b=Read(B)
Write(B,b+100)
end if
Introduction

• Transactions provide ACID properties:
  – Atomicity: All (commit) or nothing (abort).
  – Consistency: Program correctness.
  – Isolation: serializability, equivalent to a serial execution.
  – Durability: once a transaction commits its effects remain despite of failures.
Introduction

• Isolation is provided by concurrency control protocols -> e.g., locking
• Atomicity and durability are provided by recovery protocols
• Correctness criteria: different isolation levels.
Concurrency control

- If there is no concurrency control two problems may happen: lost updates and inconsistent retrievals (dirty reads).
- A lost update occurs when two transactions both read the old value of a variable and use it to calculate a new value.
- Inconsistent retrievals occur when a retrieval transaction observes values that are involved in an ongoing updating transaction.
Concurrency control: Lost updates

<table>
<thead>
<tr>
<th>bal</th>
<th>a</th>
<th>bal</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>(0)</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>(1)</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>(2)</td>
<td></td>
</tr>
<tr>
<td>700</td>
<td>(3)</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>(4)</td>
<td></td>
</tr>
</tbody>
</table>
Concurrency control: inconsistent retrievals

```
Withdraw(100)        (1)
bal= a.GetBalance()               (2)
bal = bal +b.GetBalance()     (3)
Deposit(100)            (4)
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>bal</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>-----</td>
</tr>
<tr>
<td>500</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td>400</td>
<td>700</td>
<td>700</td>
</tr>
</tbody>
</table>
Serial equivalence

• A serial equivalence interleaving is one in which the combined effect is the same effect as if the transactions have been executed sequentially in some order.

• the same effect means:
  – the read operations return the same values
  – The variables have the same values at the end

• For two transactions to be *serially equivalent*, it is necessary and sufficient that all pairs of conflicting operations of the two transactions be executed in the same order at all of the objects they both access.

• Two operations conflict if they access the same item and at least one of the operations is a write operation.
Serial equivalence

Example:

- $T$: $x = \text{read}(i); \text{write}(i, 10); \text{write}(j, 20)$;
- $U$: $y = \text{read}(j); \text{write}(j, 30); z = \text{read}(i)$;
  - serial equivalence requires that either
    - $T$ accesses $i$ before $U$ and $T$ accesses $j$ before $U$. or
    - $U$ accesses $i$ before $T$ and $U$ accesses $j$ before $T$.

- Concurrency control protocols: pessimistic (locking) and optimistic ones (backward and forward validation).
ANSI isolation levels

• ANSI isolation levels:
  – Read uncommitted.
  – Read committed.
  – Repetable read.
  – Serializable.
ANSI Isolation levels

• ANSI isolation levels are defined based on the anomalies they avoid.

• The anomalies are:
  – Dirty read:
    • T1 updates an item. T2 reads that item. T1 aborts. T2 has read an update that never happened.
  – Non-repeatable read:
    • T1 reads an item. T2 updates the same item and commits. T1 reads the same item again and it has changed its value.
  – Phantom:
    • T1 retrieves the set of items that satisfy a predicate. T2 inserts new data that satisfy that predicate and commits. T1 repeats the retrieval with the same predicate and there are new items in the result.
### ANSI Isolation levels

<table>
<thead>
<tr>
<th>Isolation Level /Anomalies</th>
<th>Dirty Write</th>
<th>Dirty Read</th>
<th>Non-repeatable Read</th>
<th>Phantom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read uncommitted</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Read committed</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Repeatable read</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Serializable</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Snapshot Isolation

• This isolation criteria avoids all ANSI anomalies.
• Oracle was the first company to implement it.
• This isolation level is defined for multiversion DBs.
• A transaction reads the most recent committed data at the time the transaction start.
• The only conflicts are write-write conflicts.
• Two concurrent transactions conflict if they update the same item. Either the first committer wins or the first transaction that updates the item.
Database Replication

• Goal: increase data availability (fault-tolerance) and efficiency.
  – Read operations can be done in any replica.
  – Local data access.

• Correctness criteria: one-copy correctness (serializability or SI). The effect of executing a set of transactions on a replicated database is equivalent to the execution of the same set of transactions over a non-replicated database.

• All copies must be updated and kept consistent.
Database replication

- Updates are propagated in the context of the original transaction -> *Eager replication* (synchronous replication).
  - No inconsistencies.
  - Increased latency.
- Updates are propagated in a different transaction -> *Lazy replication* (Asynchronous replication).
- Updates can be executed at any replica -> *Update everywhere*.
- Updates can be executed only at a given replica -> *primary copy*. Queries can be executed at any replica. Used by commercial DBs.

<table>
<thead>
<tr>
<th>Where updates are executed</th>
<th>Primary</th>
<th>Any replica</th>
</tr>
</thead>
<tbody>
<tr>
<td>When updates are propagated to other replicas</td>
<td>Eager primary copy</td>
<td>Eager update everywhere</td>
</tr>
<tr>
<td>In the original transaction</td>
<td>Lazy primary copy</td>
<td>Lazy update everywhere</td>
</tr>
</tbody>
</table>
Database replication

• Updates can be done at all the replicas at the same time or locally and at commit time the other replicas are updated (deferred updates).
  – Deferred updates need less messages. All updates can be sent in a single message (commit prepare)
  – Aborts are less expensive (only one replica needs to undo the updates).
  – Late detection of conflicts (if it implements update everywhere).
Database Replication

• If there are network partitions inconsistencies may happen.
• Quorum consensus or majority quorum: only one partition can continue processing.
  – Each replica is assigned a weight.
  – All replicas are aware of the weights of each replica.
  – A quorum is a set of nodes whose weights sum more than half of the total weight. Only a quorum component will process transactions.
  – A read quorum (RT) (writeWT) is a set of replicas that satisfy
    • 2.WT > sum all weights
    • WT+RT > sum all weights.
Database Replication

- One read operation always overlaps with a write operation.
- Two write operations always overlap.

- Each data item has a version number.
- Read operations return the most recent version number.
- Read operations cannot be executed locally.
- Write operations are executed on a write quorum. The most update version in the quorum is incremented and a new version is created in all the replicas in the quorum with that version number.
- A high number of replicas is needed to tolerate failures. One failure, three replicas. Two failures, five replicas...