CS 764: Topics in Database Management Systems
Lecture 8: Granularity of Locks

Xiangyao Yu
10/4/2020
Announcement

List of project topics updated on course website
   – Please contract the instructor if you want to discuss project topics

Proposal due on **Oct. 25**
Granularity of Locks and Degrees of Consistency
in a Shared Data Base

J. B. Gray, R. A. Lorie, E. S. Futaeva, T. L. Traiger

IBM Research Laboratory
San Jose, California

The problem of choosing the appropriate granularity (size) of lockable objects is introduced and the tradeoffs between concurrency and overhead is discussed. A locking protocol which allows simultaneous locking at various granularities by different transactions is presented. It is based on the introduction of additional locks called besides the conventional shared and exclusive mode. A proof is given of the equivalence of this protocol to a model.

First the issues of consistency is a shared environment is analyzed. This discussion is motivated by the realization that some existing data base systems use attomistic lock protocols which induce protection only from certain types of inconsistencies (for instance those arising from external dependencies) in the database. A limited degree of consistency. Four degrees of operations are introduced. They can be simply characterized as follows. Degree 0 protects shares from read updates. Degree 1 additionally provides protection from bucket updates. Degree 2 additionally provides protection from reading inconsistent data items, and degree 3 additionally provides protection from reading inconsistent relationships between data items. Full protection for all four degrees of operations would be provided by locking on the entire file. Full protection for all four degrees of locking protocols, concurrency, overhead, recovery and transaction structure.

Lastly, these ideas are compared with existing data management systems.

I: GRANULARITY OF LOCKS

An important issue which arises in the design of a data base management system is the choice of lockable units. In the context of transactional units, i.e., the data objects which are atomically locked to insure consistency, samples of lockable units are page, block, individual records, field values, and intervals of field values.

The choice of lockable units presents a tradeoff between concurrency and overhead, which is related to the size of the granularity of the site itself. On the one hand, concurrency is increased if a fine lockable unit (for example a record or field) is chosen. Such unit is appropriate for a "simple" transaction which accesses few records. On the other hand a fine unit of locking would be useless for a "complex" transaction which accesses a large number of records. Such a transaction would have to set and reset a large
Agenda

Transaction basics
Locking granularity
Two-phase locking
Degree of consistency
ACID Properties in Transactions

A sequence of many actions considered to be one atomic unit of work

**Atomicity:** Either all operations occur, or nothing occurs (all or nothing)

**Consistency:** Integrity constraints are satisfied

**Isolation:** How operations of transactions interleave

**Durability:** A transaction’s updates persist when system fails

This lecture touches A, C, and I
Locking Granularity

Use locks to prevent conflicts
Locking Granularity

Use locks to prevent conflicts

Choosing a locking granularity

– Entire database
– Relation
– Records …
Locking Granularity

Use locks to prevent conflicts

Choosing a locking granularity

- Entire database
- Relation
- Records ...

Goal: high concurrency and low cost

Increasing concurrency
Increasing overhead when many records are accessed
Locking Granularity

Use locks to prevent conflicts

Choosing a locking granularity

- Entire database
- Relation
- Records …

Goal: high concurrency and low cost

Solution: Hierarchical locks
Hierarchical Locks

Lock a high-level node if a large number of records are accessed
- All descendants are implicitly locked in the same mode
Hierarchical Locks

Lock a high-level node if a large number of records are accessed

- All descendants are implicitly locked in the same mode
- **Intention lock** to avoid conflict with implicit locks
Locking Modes

Basic locking modes
- S: Shared lock
- X: Exclusive lock
Basic locking modes
- S: Shared lock
- X: Exclusive lock

Intention modes:
- IS: Intention to share
- IX: Intention to acquire X lock below the lock hierarchy
- SIX: Read large portions and update a few parts
Locking Modes

Basic locking modes
- S: Shared lock
- X: Exclusive lock

Intention modes:
- IS: Intention to share
- IX: Intention to acquire X lock below the lock hierarchy
- SIX: Read large portions and update a few parts

Example: read record

```
<table>
<thead>
<tr>
<th>DB</th>
<th>IS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Areas</td>
<td>IS</td>
</tr>
<tr>
<td>Files</td>
<td>IS</td>
</tr>
<tr>
<td>Records</td>
<td>S</td>
</tr>
</tbody>
</table>
```
Locking Modes

Basic locking modes
- S: Shared lock
- X: Exclusive lock

Intention modes:
- IS: Intention to share
- IX: Intention to acquire X lock below the lock hierarchy
- SIX: Read large portions and update a few parts

Example: read record   update record

<table>
<thead>
<tr>
<th>DB</th>
<th>IS</th>
<th>IX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Areas</td>
<td>IS</td>
<td>IX</td>
</tr>
<tr>
<td>Files</td>
<td>IS</td>
<td>IX</td>
</tr>
<tr>
<td>Records</td>
<td>S</td>
<td>X</td>
</tr>
</tbody>
</table>
## Locking Modes

### Basic locking modes
- S: Shared lock
- X: Exclusive lock

### Intention modes:
- IS: Intention to share
- IX: Intention to acquire X lock below the lock hierarchy
- SIX: Read large portions and update a few parts

### Example:
**Example: read record**
- DB
  - IS
- Areas
  - IS
- Files
  - IS
- Records
  - S

**update record**
- IX

**scan + occasional updates**
- IX

**lock specific records in X mode**

```plaintext
<table>
<thead>
<tr>
<th>Levels</th>
<th>IS</th>
<th>IX</th>
<th>SIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td></td>
<td></td>
<td>IX</td>
</tr>
<tr>
<td>Areas</td>
<td>IS</td>
<td></td>
<td>IX</td>
</tr>
<tr>
<td>Files</td>
<td>IS</td>
<td></td>
<td>SIX</td>
</tr>
<tr>
<td>Records</td>
<td>S</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
```
Example

a) [10 points] Consider the following locking hierarchy where there is a single database that contains a single table and the table contains two tuples: A and B. If a transaction T1 reads tuple A and writes tuple B, what lock modes (e.g., NL, S, X, IS, IX, SIX) will T1 hold on the tuples, the table, and the database, respectively?
Example

a) [10 points] Consider the following locking hierarchy where there is a single database that contains a single table and the table contains two tuples: A and B. If a transaction T1 reads tuple A and writes tuple B, what lock modes (e.g., NL, S, X, IS, IX, SIX) will T1 hold on the tuples, the table, and the database, respectively?
Example

a) [10 points] Consider the following locking hierarchy where there is a single database that contains a single table and the table contains two tuples: A and B. If a transaction T1 reads tuple A and writes tuple B, what lock modes (e.g., NL, S, X, IS, IX, SIX) will T1 hold on the tuples, the table, and the database, respectively?
## Lock Compatibility

### Increasing lock strength

<table>
<thead>
<tr>
<th></th>
<th>IS</th>
<th>IX</th>
<th>S</th>
<th>SIX</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>IX</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>S</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SIX</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>X</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

### Most privileged

Most privileged

### Least privileged

Least privileged
## Lock Compatibility

### Increasing lock strength

<table>
<thead>
<tr>
<th></th>
<th>IS</th>
<th>IX</th>
<th>S</th>
<th>SIX</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>IX</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>S</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SIX</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>X</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

### Most privileged

- X
- SIX
- IS
- NL

### Least privileged

- S
- IX
Rules for Lock Requests

• Before requesting S or IS on a node, all ancestor nodes of the requested node must be held in IS or IX
Rules for Lock Requests

• Before requesting S or IS on a node, all ancestor nodes of the requested node must be held in IS or IX

• Before requesting X, SIX, or IX on a node, all ancestor nodes of the requesting node must be held in SIX or IX
Rules for Lock Requests

• Before requesting S or IS on a node, all ancestor nodes of the requested node must be held in IS or IX

• Before requesting X, SIX, or IX on a node, all ancestor nodes of the requesting node must be held in SIX or IX

• Locks requested root to leaf
• Locks released leaf to root or any order at the end of the transaction (as an atomic operation)
Extension – Semantic Locking

A system can introduce new lock types based on the operation semantics
Extension – Semantic Locking

A system can introduce new lock types based on the operation semantics.

Example:

- Increment and decrement values

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>INC</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>INC</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>X</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Extension – Semantic Locking

A system can introduce new lock types based on the operation semantics

Example:
- Increment and decrement values
- Test value is greater than X

Example: increment lock

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>INC</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>INC</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>X</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Example: compare with constant

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>COMP</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>COMP</td>
<td>Y</td>
<td>Y</td>
<td>depends</td>
</tr>
<tr>
<td>X</td>
<td>N</td>
<td>depends</td>
<td>N</td>
</tr>
</tbody>
</table>
To avoid starvation (where a transaction is delayed indefinitely), each request waits its turn in the queue.
Deadlock

tuple A

\[ \text{T1.S} \rightarrow \text{T2.X} \quad \# \text{T2 waits for T1} \]

tuple B

\[ \text{T2.S} \rightarrow \text{T1.X} \quad \# \text{T1 waits for T2} \]
Deadlocks Solutions

**Deadlock detection**: Once a cycle is detected, abort a transaction in the cycle
Deadlocks Solutions

**Deadlock detection**: Once a cycle is detected, abort a transaction in the cycle

**No-Wait**: A transaction self-aborts when encountering a conflict
Deadlocks Solutions

**Deadlock detection**: Once a cycle is detected, abort a transaction in the cycle

**No-Wait**: A transaction self-aborts when encountering a conflict

**Wait-Die**: On a conflict, the requesting transaction *waits* if it has higher priority than transactions in the queue, otherwise the requesting transaction *self-aborts*
Deadlocks Solutions

**Deadlock detection**: Once a cycle is detected, abort a transaction in the cycle

**No-Wait**: A transaction self-aborts when encountering a conflict

**Wait-Die**: On a conflict, the requesting transaction waits if it has higher priority than transactions in the queue, otherwise the requesting transaction self-aborts

**Wound-Wait**: On a conflict, the requesting transaction preemptively aborts current owners if it has higher priority, otherwise the requesting transaction waits
Serializability

Concurrent execution of transactions produces the same results as some serial execution

- Intuitive and easy to reason about
Two-Phase Locking (2PL)

**Two-phase locking** (2PL) ensures serializability

- Growing phase: acquiring locks (no release)
- Shrinking phase: releasing locks (no acquire)
Two-Phase Locking (2PL)

Two-phase locking (2PL) ensures serializability
- Growing phase: acquiring locks (no release)
- Shrinking phase: releasing locks (no acquire)
- Serialization point: after all locks are acquired but before any release
- The equivalent serial order = order of transactions’ serialization points
Two-Phase Locking (2PL)

**Two-phase locking** (2PL) ensures serializability
- Growing phase: acquiring locks (no release)
- Shrinking phase: releasing locks (no acquire)
- Serialization point: after all locks are acquired but before any release
- The equivalent serial order = order of transactions’ serialization points

**Strict 2PL**: 2PL + all exclusive locks released after transaction commits
- Widely used scheme in practice
Degree of Consistency (Isolation)

Degree 3: Serializability (assuming no phantom effect)
   – Two-phase with respective to both reads and writes
Degree of Consistency (Isolation)

Degree 3: Serializability (assuming no phantom effect)
- Two-phase with respective to both reads and writes

Degree 2: Read Committed
- Two-phase with respect to writes
- Short read locks
Degree of Consistency (Isolation)

Degree 3: Serializability (assuming no phantom effect)
  – Two-phase with respective to both reads and writes

Degree 2: Read Committed
  – Two-phase with respect to writes
  – Short read locks

Degree 1: Read Uncommitted
  – Two-phase with respect to writes
  – No read locks (may observe dirty data)
Degree of Consistency (Isolation)

Degree 3: Serializability (assuming no phantom effect)
  – Two-phase with respective to both reads and writes

Degree 2: Read Committed
  – Two-phase with respect to writes
  – Short read locks

Degree 1: Read Uncommitted
  – Two-phase with respect to writes
  – No read locks (may observe dirty data)

Degree 0:
  – Short write locks
  – No read locks
Degree of Consistency (Isolation)

Degree 3: Serializability (assuming no phantom effect)
- Two-phase with respective to both reads and writes

Degree 2: Read Committed
- Two-phase with respect to writes
- Short read locks

Degree 1: Read Uncommitted
- Two-phase with respect to writes
- No read locks (may observe dirty data)

Degree 0:
- Short write locks
- No read locks
Q/A – Granularity of Locks

Optimal schedule based on knowledge of the workload?
Intention locks used today?
Phantom effect?
Paper hard to follow…
Submit review for