CS 764: Topics in Database Management Systems
Lecture 9: Granularity of Locks

Xiangyao Yu
10/5/2022
Announcement

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

Two types of projects in general
  – Survey/Evaluation
  – Research projects

Proposal due on Oct. 24
Announcement

Project proposal deadline: **Oct. 24**

Make sure to cover the following aspects (in 1 or 2 pages)

- Project name
- Author list
- Background and motivation (why is the problem important? what are the challenges)
- Task plan (what will you do in the project? what are your key contributions?)
- Timeline

Submission website: [https://wisc-cs764-f22.hotcrp.com](https://wisc-cs764-f22.hotcrp.com)

Recommend ACM format

- [https://www.acm.org/publications/proceedings-template](https://www.acm.org/publications/proceedings-template)
Granularity of locks and degrees of consistency in a shared data base

J. E. Gray, B. A. Lorie, C. S. Patilas, L. L. Wegman
IBM Research Laboratory
San Jose, California

The problem of choosing the appropriate granularity (size) of lockable objects is introduced and the tradeoff between concurrency and overhead is discussed. A locking protocol which allows maintenance looking at various characteristics of different transactions is presented. It is based on the introduction of additional lock modes besides the conventional share and exclusive modes. Proof is given of the equivalence of this protocol to a conventional protocol.

Part the issue of consistency in a shared environment is analyzed. This discussion is motivated by the realization that some existing data base systems use automatic lock protocols which impose protection only from certain types of inconsistencies. For instance, those aiming for single record consistency. Three lock modes are introduced: a single lock mode for a limited degree of consistency. Four degrees of consistency are introduced. They can be simply characterized as follows: degree 0 protects atoms from four updates. Degree 1 additionally provides protection from blocking updates. Degree 2 additionally provides protection from reading incorrect data, and degree 3 additionally provides protection from reading incorrect data and from reading incorrect relationships among data items. In practice, these lock modes provide the relationships of the four degrees to locking protocols, concurrency, overhead, recovery, and transaction structure.

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

I. DEGREES OF LOCKS

An important issue which arises in the design of a data base management system is the choice of lockability units. I.e. the data aggregates which are logically linked to ensure consistency. Examples of lockable units are a row, a line, or individual records, field values, and interval of field values.

The choice of lockable units presents a tradeoff between concurrency and overhead, which is balanced in the size of lockable units. In 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 "single" transaction which accesses few records. On the other hand, finer locking would make for a "complex" transaction which accesses a large number of records. Such a transaction would have to wait and resend a large number of records. Each of these issues would limit the lockable unit size.
Agenda

Transaction basics
Locking granularity
Two-phase locking
Degree of consistency
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
Agenda

Transaction basics

**Locking granularity**

Two-phase locking

Degree of consistency
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
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

```
DB  |  IS
   |    
Areas | IS
   |    
Files | IS
   |    
Records | S
```
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></th>
<th>IS</th>
<th>IX</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>IS</td>
<td>IX</td>
<td></td>
</tr>
<tr>
<td>Areas</td>
<td>IS</td>
<td>IX</td>
<td></td>
</tr>
<tr>
<td>Files</td>
<td>IS</td>
<td>IX</td>
<td></td>
</tr>
<tr>
<td>Records</td>
<td>S</td>
<td>X</td>
<td></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, scan + occasional updates

<table>
<thead>
<tr>
<th>DB</th>
<th>Areas</th>
<th>Files</th>
<th>Records</th>
<th>read record</th>
<th>update record</th>
<th>scan + occasional updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>IS</td>
<td>IS</td>
<td>S</td>
<td>IX</td>
<td>IX</td>
<td>IX</td>
</tr>
</tbody>
</table>

- **SIX** in Records: lock specific records in X mode
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?
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 lock is `SIX`.

#### Least privileged

Least privileged lock is `X`.

---

Increasing lock strength from top to bottom:
- `IS` → `IX` → `S` → `SIX` → `X`
## 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

\[ X \]
\[ \text{SIX} \]
\[ \text{IX} \]
\[ \text{S} \]
\[ \text{IS} \]
\[ \text{NL} \]

### Least privileged
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 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:
- Increment and decrement values
### 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 \( V \)

#### 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
T1.S — T2.X  # T2 waits for T1

tuple B
T2.S — T1.X  # 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
Agenda

Transaction basics
Locking granularity
Two-phase locking
Degree of consistency
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
Agenda

Transaction basics
Locking granularity
Two-phase locking

Degree of consistency
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

Increasing concurrency
Weaker guarantees
What degree of consistency do modern systems adopt?

Can we leverage the workload information to better schedule txns?

Locking all ancestors up to the root introduce overhead?

Can we downgrade the lock mode?

What are phantom effects?
Submit review for