Distributed Transaction Processing with MySQL XA

Mark Matthews, MySQL AB

MySQL Users Conference, 2005
April 18-21, 2005
MySQL XA: Session Outline

• The problem:
  – What is Distributed Transaction Processing?
  – Why “distributed transactions”? 
  – X/Open DTP model
  – XA interface

• Solution:
  – MySQL XA
  – Java Transactional API

• MySQL “internal XA”
Who Developed our XA impl?

- Sergei Golubchik
- Senior Software Developer in MySQL AB
- Originally from Ukraine, now living in Osnabrück, DE
- Working on MySQL server internals since 1998
- Full-time MySQL AB employee since March 2000
- Main project — Fulltext Search, but also during these years worked with almost every part of MySQL server source
What is DTP

• Distributed Transaction Processing
• Maintaining consistency between several separated databases
• Example: wire transfer
  • Bank 1:
    – UPDATE account SET balance=balance – 100000
      WHERE account_nr=12345678
  • Bank 2:
    – UPDATE account SET balance=balance + 100000
      WHERE account_nr=9876543
How to do it wrong

• Bank 1:
  – UPDATE account SET balance=balance – 100000
    WHERE account_nr=12345678
  – COMMIT

• Bank 2:
  – UPDATE account SET balance=balance + 100000
    WHERE account_nr=9876543 → ERROR

• Bank 1:
  – Oops
How to do it wrong

- Bank 1:
  - UPDATE account SET balance=balance – 100000 WHERE account_nr=12345678
- Bank 2:
  - UPDATE account SET balance=balance + 100000 WHERE account_nr=9876543
  - COMMIT
- Bank 1:
  - ERROR (e.g. HD crash, power failure, or alien attack)
- Bank 2
  - Oops
Two-Phase Commit

- Bank 1:
  - UPDATE account SET balance=balance – 100000 WHERE account_nr=12345678
  - PREPARE

- Bank 2:
  - UPDATE account SET balance=balance + 100000 WHERE account_nr=9876543
  - COMMIT

- Bank 1:
  - COMMIT
X/Open DTP Model

• An Application Program (AP)
  – defines transaction boundaries
  – specifies actions that constitute a transaction
• Resource Managers (RM's)
  – provide access to shared resources
• A Transaction Manager (TM)
  – assigns identifiers to transactions
  – takes responsibility for transaction completion and recovery
X/Open DTP Model

Application Program (AP)

Resource Managers (RM's)

START TRANSACTION, COMMIT, ROLLBACK

Two-Phase Commit

INSERT, UPDATE, DELETE, SELECT,...
X/Open DTP Model

Application Program (AP)

Resource Managers (RM's)

Transaction Manager (TM)

START TRANSACTION, COMMIT, ROLLBACK

INSERT, UPDATE, DELETE, SELECT, ...

START TRANSACTION

Two-Phase Commit

XA
**XA Interface: XID**

- **Transaction/Branch Identifier:**
  - **GTRID:** Global Transaction ID — max. 64 bytes
    - Identifies global transaction — atomic unit of work
  - **BQUAL:** Branch Qualifier — max. 64 bytes
    - Identifies transaction branch within global transaction
    - Each branch participates separately in two-phase commit protocol
  - XID's must be globally unique
### XA Routines

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ax_reg</td>
<td>Register an RM with a TM.</td>
</tr>
<tr>
<td>ax_unreg</td>
<td>Unregister an RM with a TM.</td>
</tr>
<tr>
<td>xa_close</td>
<td>Terminate the AP's use of an RM.</td>
</tr>
<tr>
<td>xa_commit</td>
<td>Tell the RM to commit a transaction branch.</td>
</tr>
<tr>
<td>xa_complete</td>
<td>Test an asynchronous xa_ operation for completion.</td>
</tr>
<tr>
<td>xa_end</td>
<td>Dissociate the thread from a transaction branch.</td>
</tr>
<tr>
<td>xa_forget</td>
<td>Permit the RM to discard its knowledge of a heuristically-completed transaction branch.</td>
</tr>
<tr>
<td>xa_open</td>
<td>Initialise an RM for use by an AP.</td>
</tr>
<tr>
<td>xa_prepare</td>
<td>Ask the RM to prepare to commit a transaction branch.</td>
</tr>
<tr>
<td>xa_recover</td>
<td>Get a list of XIDs the RM has prepared or heuristically completed.</td>
</tr>
<tr>
<td>xa.rollback</td>
<td>Tell the RM to roll back a transaction branch.</td>
</tr>
<tr>
<td>xa_start</td>
<td>Start or resume a transaction branch — associate an XID with future work that the thread requests of the RM.</td>
</tr>
</tbody>
</table>
Anatomy of the distributed transaction

- 2. TM → RM1: xa_start
- 1. TM → RM2: xa_start
- 0. the real work is here
- 1. TM → RM1: xa_prepare; RM1: record transaction
- 2. TM → RM2: xa_prepare; RM2: record transaction
- 3. TM: record commit decision
- 4. TM → RM1: xa_commit; RM1: commit
- 5. TM → RM2: xa_commit; RM2: commit
- 6. done!
Crash Recovery

- TM → RM1: xa_recover
- RM1 → TM: xid1, xid2, xid5, ...
- TM → RM2: xa_recover
- RM2 → TM: xid1, xid3, xid4, ...
- TM looks in the log: xid1, xid2, xid6 ...
Crash Recovery

- TM → RM1: xa_recover
- RM1 → TM: xid1, xid2, xid5, ...
- TM → RM2: xa_recover
- RM2 → TM: xid1, xid3, xid4, ...
- TM looks in the log: xid1, xid2, xid6 ...
- TM → RM1: xa_commit(xid1), xa_commit(xid2), xa_rollback(xid5)
- TM → RM2: xa_commit(xid1), xa_rollback(xid3), xa_rollback(xid4)
Anatomy of the distributed transaction

-2. TM → RM1: xa_start
-1. TM → RM2: xa_start
  0. the real work is here
  1. TM → RM1: xa_prepare; RM1: record transaction (sync)
  2. TM → RM2: xa_prepare; RM2: record transaction (sync)
  3. TM: record commit decision (sync)
  4. TM → RM1: xa_commit; RM1: commit (sync)
  5. TM → RM2: xa_commit; RM2: commit (sync)
  6. done!
Optimizing 2PC: Lazy xa_start

• AP → TM: “start transaction”
  – TM: does nothing
• AP → RM1 (via TM): “do something”
  – TM → RM1: xa_start
  – TM1 → RM1: “do something”
• AP → RM2 (via TM): “do something”
  – TM → RM2: xa_start
  – TM1 → RM2: “do something”
Optimizing 2PC: Fallback to 1PC

-2. TM → RM1: xa_start
-1. TM → RM2: xa_start
0. the real work is here
1. TM → RM1: xa_prepare; RM1: record transaction (sync)
2. TM → RM2: xa_prepare; RM2: record transaction (sync)
3. TM: record commit decision (sync)
4. TM → RM1: xa_commit; RM1: commit (sync)
5. TM → RM2: xa_commit; RM2: commit (sync)
6. done!
Optimizing 2PC: Fallback to 1PC

-2. TM → RM1: xa_start
-1. TM → RM2: xa_start
0. the real work is here
1. TM → RM1: xa_prepare; RM1: record transaction (sync)
2. TM → RM2: xa_prepare; RM2: record transaction (sync)
3. TM: record commit decision (sync)
4. TM → RM1: xa_commit; RM1: commit (sync)
5. TM → RM2: xa_commit; RM2: commit (sync)
6. done!
Optimizing 2PC: Fallback to 1PC

-2. TM → RM1: xa_start

0. the real work is here

1. TM → RM1: xa_prepare; RM1: record transaction (sync)

3. TM: record commit decision (sync)

4. TM → RM1: xa_commit; RM1: commit (sync)

6. done!
Commit Grouping

• My home PC: 1,000 sync/sec
• For TM it means: no more than 1,000 commits/sec
• If commits are coming at rate 5,000/sec, then while the first will be syncing, 5 new will come.
• Let's log them in one I/O block and sync in one do.
• Throughput with grouping: 1,000*xids_per_io_block
• For MySQL as TM it means 680,000 commits/sec
• But don't take this number literally! 😊
OS-specific tricks

• TM has to fsync() but it does not have to write()
• By mapping the complete log into the memory with mmap(), one can sync twice as fast
• My home PC:
  – write()+fsync() → ~ 300–500 syncs/sec
  – mmap()+fsync() → ~ 1000 syncs/sec
• fdatasync() instead of fsync()
XA inside MySQL

- MySQL has many storage engines
- You can use more than one storage engine in one transaction
- Therefore MySQL needs 2PC even internally to guarantee consistency
- MySQL acts as a TM, with storage engines being RM's. MySQL is responsible for recovery logging
- It is implemented in 5.1
- All mentioned optimizations are, of course fully implemented, and more are to come...
MySQL XA Statements

- XA START xid  [ JOIN | RESUME ]
- XA END xid  [ SUSPEND [ FOR MIGRATE ]]
- XA PREPARE xid
- XA COMMIT xid  [ ONE PHASE ]
- XA ROLLBACK xid
- XA RECOVER
2PC Example

- 1> XA START 0xdb4ecb,0x65785841,48801;
- 2> XA START 0xdb4ecb,0x65785841,48801;
- 1> SELECT * FROM stats;
- 2> INSERT INTO aggr VALUES (...);
- 1> DELETE FROM stats;
- 1> XA PREPARE 0xdb4ecb,0x65785841,48801;

- 2> XA COMMIT 0xdb4ecb,0x65785841,48801;
- 1> XA COMMIT 0xdb4ecb,0x65785841,48801;
XA Primer with JTA

• The “2PC example” was too simple
• Does not follow X/Open DTP Model
• No TM to perform recovery logging — no recovery in case of crash
• Using MySQL XA statements one can implement complete TM relatively easy
• When using Java Transaction API one usually has a TM available
Questions ?