Distributed Systems

Seminar 5 – Distributed Transactions

Dr. Thomas Springer
Exercise 5.1

Foundations

a) State the fundamental properties of a transaction. What mechanisms are required to ensure these properties in distributed transactions?

b) What extensions provide a distributed transaction in comparison to a local data base transaction? What additional mechanisms are required to handle distributed transactions?
Transactions implement the **ACID**-criteria:

**Atomicity**
- Execution complete or without impacts

**Consistency**
- Transformation between consistent states

**Isolation**
- No overlapping of concurrent transaction executions

**Durability**
- Survival of system failures due to persistent storage
What extensions provide a distributed transaction in comparison to a local data base transaction? What additional mechanisms are required to handle distributed transactions?

Distributed Transaction:
- Integrates operations on objects/data residing on multiple servers into an atomic operation/transaction

Additions/Extensions
- Coordination of multiple servers
- Distributed Concurrency Control/Locking mechanisms
  - Distributed deadlock handling
- Handling of transaction context in distributed manner
- Distributed transaction completion – commit protocol
Ensuring ACID properties (Atomicity, Consistency, Isolation and Durability):

- Either both Warehouse Management and the Dispatch list are updated or non at all
- Sum of products in the Dispatch List must be consistent with that in Warehouse Management
- Modifications to number of products are not allowed to become visible for other operations outside of the transaction until Warehouse Management and Dispatch are both updated and transaction successfully closed.
The Two-Phase-Commit protocol is employed for the conclusion of distributed transactions.

a) Illustrate by means of a sequence diagram the timely sequence of an error free communication.

b) How does the protocol react to the failure of the coordinator after sending the “Prepare” message?

c) How does the protocol react to the failure of a participant, when this participant has already sent a “Ready” message and all other participants as well as the coordinator have correspondingly replied with “Ready”? 
Solution E5.2a

Diagram:

- Teilnehmer 1
  - Phase 1: Prepare → Ready → Commit → Done
  - Phase 2: Done → Commit → Ready → Prepare

- Koordinator
  - Prepare → Ready → Commit → Done

- Teilnehmer 2
  - Phase 1: Prepare → Ready → Commit → Done
  - Phase 2: Done → Commit → Ready → Prepare

Period of uncertainty with the participants.
How does the protocol react to the failure of the coordinator after sending the “Prepare“ message?

- At Participants timeout after sending “Ready/Abort“
- Restart coordinator
- Coordinator recovers transaction state, resends “Prepare“
- Participants answer

Period of uncertainty

- Before commit message is received locks can not be released
- Blocking of data access in phase of uncertainty
- Resources for Transaction (Log, data copies, state) can not be released

Also if only one participant fails after receiving „Prepare“ but before sending „Ready/Abort“
If a participant fails after having sent a “Ready” message, the

- coordinator can successfully conclude the transaction with a “Commit” message if all other participants have already answered with “Ready” and everything ran successfully with the coordinator.
- Thereupon, the coordinator sends a “Commit” message which can successfully conclude the transaction, release locks and discard intermediate processing results. A “Rollback” is therewith no longer possible.
- The failed participant now ascertains after a restart that the transaction has not yet been concluded and a “Commit” or “Rollback” ordered by the coordinator is still outstanding.
- The participant therefore asks the coordinator what the current state of the transaction is.
- The coordinator states that the transaction was successfully concluded, whereupon the participant can then conclude the transaction to agree with this.
Concurrency control is essential to ensure the isolation of transactions.

a) Clarify the terms pessimistic and optimistic concurrency control.

b) Sketch the setting and release of locks for transaction T1 in the figure below according to the simple and strict Two-Phase-Locking (TPL) algorithms. It is assumed that the system uses write locks only and read and write operations have a duration of 1 time unit.

c) From what point in time on, a second transaction T2 could perform the operations r(x) and w(z,5) for simple and strict 2PL.
Solution E5.3a

- **Trade-off**
  - effort for rollback and parallel access to data

- **Pessimistic concurrency control**
  - Assumption frequent conflicts and costly rollbacks -> avoid rollbacks
  - Locking - i.e. data is reserved exclusively for a particular transaction
  - Other transactions must wait until release of locks to access same data – no parallel processing of locked data

- **Optimistic concurrency control**
  - Assumption rare conflicts and rollbacks -> allow concurrency
  - No locks set – parallel processing of locked data allowed
  - Cascading rollback could be required
  - Compensation operations instead of rollback
Common interactions between lock types

- Existing *read-lock* blocks intended *write* but does not block further *read-locks*
- Existing *write-lock* blocks intended *read* and *write* on same object

**X** indicates incompatibility, i.e. multiple locks can’t be established in parallel on the same object

<table>
<thead>
<tr>
<th>Lock type</th>
<th>read-lock</th>
<th>write-lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>read-lock</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>write-lock</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

It is assumed that the system uses write locks only and read and write operations have a duration of 1 time unit.
• **T1 - Simple 2PL**
  - \( L(x) = t_2 \)
  - \( L(y) = t_5 \)
  - \( L(z) = t_{11} \)
  - \( U(x) = t_{12} \)
  - \( U(z) = t_{12} \)
  - \( U(y) = t_{16} \)
• **T1 - Strict 2PL**
  - \( L(x) = t_2 \)
  - \( L(y) = t_5 \)
  - \( L(z) = t_{11} \)
  - \( U(x) = t_{20} \)
  - \( U(z) = t_{20} \)
  - \( U(y) = t_{20} \)
• **T2 - Simple 2PL**
  - \( R(x) = t_{12} \)
  - \( W(y, 5) = t_{16} \)
• **T2 – Strict 2PL**
  - \( R(x), W(y, 5) = t_{20} \)

**Issue**
Simple 2PL – more parallelism but cascading rollbacks possible, avoided with strict 2PL
The reference model of the Open Group for Distributed Transaction Processing (DTP) introduces Resource Managers, Application Programmes and Transaction Monitors as main components for distributed Transactions.

a) Name the main responsibilities of these components.

b) What operations are declared in the TX and XA interfaces?

c) Sketch the necessary components and message exchange according to the DTP reference model to implement the Product Order Processing as described in slide 5.8 of the lecture.
- **Application Programme (AP)**
  - Executes operations on various resources involved within a distributed transaction
  - Defines transaction boundaries and specifies actions that constitute a transaction.
  - Defines begin and end of transactions
  - Decides whether a transaction is concluded with a “commit” or “rollback”

- **Resource Manager (RM)**
  - Represents resources involved within a distributed transaction e.g. databases, print servers etc.
  - Provide access to shared resources
  - Executes local transactions on these resources
  - Coordinates conclusion of such transactions

- **Transaction Manager (TM)**
  - Manages execution and coordination of transaction conclusion with various Resource Managers
  - Achieved on the basis of the 2 Phase-Commit-Protocol
  - Executes commit and reset operations
Solution E5.4b

- **TX interface** – between Application Programme and Transaction Manager
  - `tx_open, tx_close`
  - connect to transaction manager
  - `tx_begin, tx_rollback, tx_commit`
  - demarcate transaction
- XA interface – between Transaction Manager and Resource Manager
  - `xa_open, xa_close`
  - initialization before any other call, termination after all other calls
  - `xa_start, xa_end`
  - associate subsequent calls with a transaction branch
  - `xa_prepare, xa_commit, xa_rollback`
  - classical two phase commit protocol

- `ax_reg`
  - Register an RM with a TM.
- `ax_unreg`
  - Unregister an RM with a TM.
Solution E5.4c

**Overview:**
- **Initiation:**
  - AP initiates transaction by calling `xa_open(RM1, RM2)`.
  - TM and RM1 acknowledge the open request.
- **Transaction:**
  - AP calls `xa_start()`.
  - TM and RM1 acknowledge the start request.
  - AP calls `requestProduct(Pid)`.
  - AP calls `returnProduct(Pid)`.
  - AP calls `tx_commit()/tx_rollback()`.
- **Two-Phase-Commit:**
  - AP calls `xa_prepare()`.
  - AP calls `xa_ready()/xa_abort()`.
  - AP calls `xa_commit()/xa_rollback()`.
  - AP calls `xa_end()`.
- **Termination:**
  - AP calls `xa_close()`.
  - TM and RM1 acknowledge the close request.

**Key Points:**
- The diagram illustrates the phases of a distributed transaction, including initiation, transaction, two-phase commit, and termination.
- The interactions are shown between AP, TM, and RM, highlighting the sequence of calls and responses.
- The diagram is a visual representation of transaction management in a distributed system.
Nested Transactions

a) Which advantages do nested distributed transactions possess over simple transactions?

b) Why must, in the case of nested distributed transactions, the locks of concluded partial transactions be kept in place until the conclusion of the whole encompassing transaction?
Nested distributed transactions allow the fixing of partial results of transactions and therewith a fine-grained resetting in the event of failures.

- This is meaningful above all when transactions take a long time, and as the case may be contain complex operations, in order to retain the results of these operations as far as possible.
- Partial results hard to achieve (book flight)

Moreover, partial transactions can be executed in parallel.

- The same operations within a flat transaction would be processed sequentially
In the case of nested distributed transactions there still exists the possibility of a rollback for concluded partial transactions, perhaps due to a failure in another partial transaction which can cause the rollback of the whole encompassing transaction.

If locks are not upheld through to the conclusion of the whole encompassing transaction, other transactions could access the results of the concluded partial transactions of other nested transactions and then also modify these results.

If successfully concluded partial transactions then have to be rolled back, this can lead to a cascade of rollbacks, meaning that the property of isolation would not be fulfilled.