A Crash Course in Recovery

Guy Pardon, Atomikos
guy@atomikos.com

Outline

- The Atomikos Perspective: About Us
- Motivation: Recoverability (What and Why)?
- The Foundation: Two-Phase Commit (2PC)
- Connector Vendor Perspective
  - The XA Protocol
- User Perspective
  - Appserver Configuration
- Conclusion

The Atomikos Perspective

Who Are We?

About Atomikos

- Greek name
  - Meaning: all-or-nothing
  - No half work (closely related to recoverability)
- Background
  - Transaction PhD research (1997-2000, ETH Zurich)
    - composite transaction model: transactions across web-services (avant la lettre)
    - prototype and theory published
    - presented at VLDB
    - patented
  - Spin-off resulting in Atomikos (early 2001)

Our Experience: Transactions

- "Tuxedo/CICS for Java"
- JTA (Java Transaction API) implementation
  - meaning: it gives you recoverability and data validity
  - edition 2.0
  - integrates with J2SE and J2EE applications
  - integrates with JBoss
- Enable recoverability:
  - across different back-end systems
  - across LAN networks
  - future: across SOAP networks
- Principle: allow incomplete tasks to be cancelled at any time
  - crash resilience: data consistency is restored after restart or crash

Motivation

Why Recoverability?

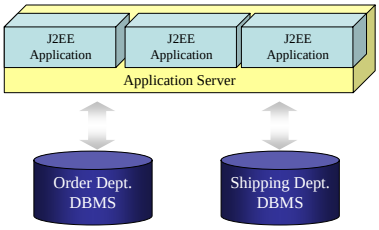
What is Recoverability?

- The ability to recover (restore) application-level data to a valid state
  - After a failure in a running system, or after a crash or restart
  - For this talk: in a distributed transaction setting
- By enforcing each individual transaction to be valid by itself
  - The data state is the sum of all past transactions!
  - So validity begins at the level of each individual transaction
- Valid transaction = all-or-nothing effects (no half work)
- This is not trivial if more than 1 back-end is accessed
  - Requires each data store to be recoverable, AND
  - Requires appserver support as well

Driving Factors

- Penetration of OS in enterprise
  - Multipe data source integration requires recovery capabilities
- Success of JMS as invocation means
  - Requires reliable processing between queue and DBMS
  - In turn requires recoverability
- Recoverability is not well understood
  - Many connector-level deviations from standards
  - Notoriously obscure specs
- Not included in JBoss by default

JDBC Example



Keeping two or more DBMS consistent requires recoverability!

JMS Example

Ensuring exactly-once requires recoverability!

The Foundation

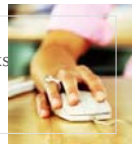
Two-Phase Commit (2PC):
Ensuring Validity of Each Transaction

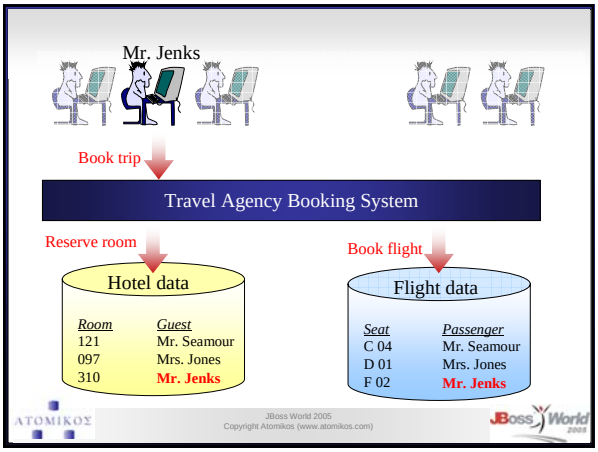
2PC Essentials

- The recoverability problem illustrated
- The solution
- Logging requirements

The Problem: Distributed Transactions

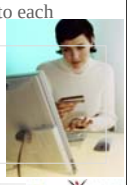
- One user transaction that accesses/updates data in more than one system
  - 'Global' transaction vs. 'local' transactions
- Is not trivial w.r.t. atomicity (all-or-nothing)
  - And therefore also w.r.t. recoverability
- Atomicity (validity) requires:
  - If global tx succeeds then so do ALL local txs
  - If global tx cancels then so do ALL local txs
  - Nothing in between!





Validity=Atomicity

- The problem:
  - If a room is booked, the flight should be OK too!
  - If a flight is booked, there should be a hotel to stay!
- Naive approach: commit ordering
  - In what order should commit be communicated to each database?



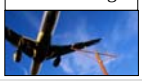
First attempt

Commit Hotel

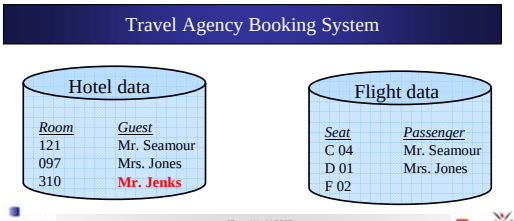
OK? — no → Rollback Flight

yes

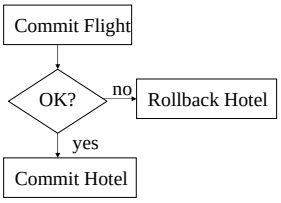
Commit Flight



Not Atomic!



Second attempt



- Same problem as in previous case
- Only the roles of hotel and flight are exchanged

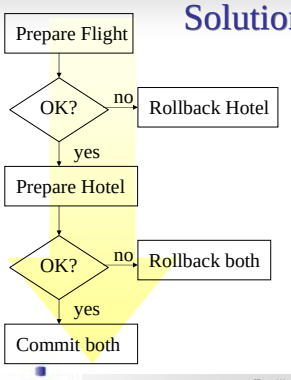
Observation

- Sending 'commit' to a database does not guarantee it will do so
  - Intermediate timeout in database -> automatic rollback!
  - Failure before commit arrives -> restart will rollback!
- But if it does, then there is no way to cancel
- Consequently, asking more than one database to commit is always a risk
  - One may fail
  - The other may succeed
  - Leading to inconsistency

The 2PC Trick

- An intermediate state (prepared state):
  - A database that is prepared can still rollback
  - But not on its own initiative!
    - No rollback on restart
    - No rollback on timeout
  - A database that is prepared can also still commit
    - Even after a crash!
  - The prepared state is like a 'checkpoint' from where both rollback and commit are possible for every database

Solution: 2PC



- First prepare each database!
- Prepare OK means: no rollback on its own
- Still allows rollback when asked for!
- Guarantees commit successful
- This is a system-level protocol between:
  - Appserver (transaction manager module)
  - Connectors to back-ends

Recoverability Requires Logging

- A DMBS has to remember prepared transactions
  - to avoid unilateral rollback after crash/restart
  - requires disk write before acknowledging prepare
- A transaction manager has to remember prepared transactions
  - to know what databases are prepared
- A transaction manager has to remember commit/rollback decisions!
  - right before a crash, it might have instructed part of the system to commit
  - after a crash, the remaining DBMS need to do the same!
  - only possible if the transaction manager supports logging

Connector Vendor Perspective

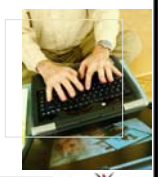
The XA Protocol

The XA Protocol

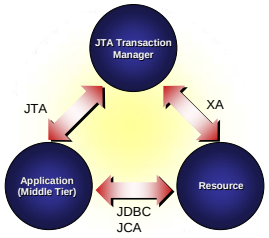
- X/Open standard specification
- Regulates interactions between a resource and a transaction manager
  - two-phase commit, but also
  - multiplexing different transactions work (SQL) over the same connection and
  - multiplexing 2PC messages and other transactional work
  - recovery after restart or crash

XA Operations

- Defined on each connection to the back-end
- Multiplexing work (SQL):
  - start
  - end
- Two-phase commit:
  - prepare
  - commit
  - rollback
- Recovery:
  - isSameRM
  - recover
  - (forget)

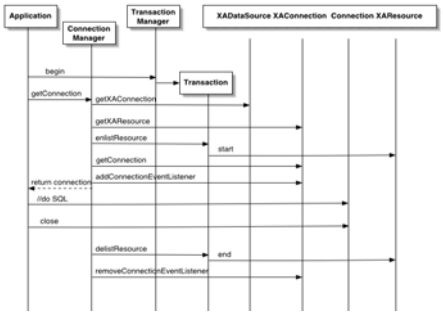


XA in Java: JTA/XA

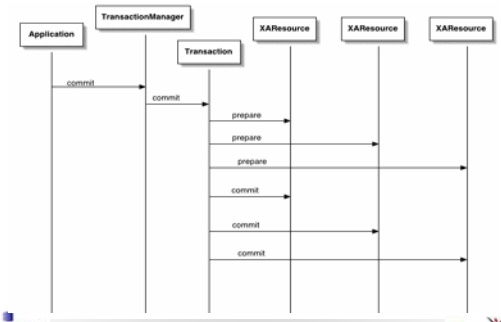


- Application-level: JTA
  - begin tx
  - commit/rollback
  - associated with running thread
- Resource-level: XA
  - prepare
  - commit
  - rollback
  - recover

JTA/XA Interactions



Commit



Connector Vendor Pitfall #1

- 2PC methods are allowed at all times
- Even if the connection has started work for another transaction already!!!
- In other words:
  - the appserver can reuse the connection immediately after end, without waiting for 2PC of the previous transaction!
- Risk of not being compliant:
  - pooling is less efficient (reuse only possible after commit)
  - pooling requires appserver workaround

Recovery

- When an XAResource is used for the first time (counted from startup of the TM), it may have to be recovered
- This will typically only be done when necessary (i.e. when first added)
- Also, what if two XAResources are for the same DBMS? This could lead to anomalies in recovery
- Ergo: the transaction manager needs a way to distinguish between new and known resources
- Enters isSameRM()!!!

Connector Vendor Pitfall #2

- Two XAResource instances should return true when compared with isSameRM(), if and only if:
  - They both return the same Xid[] when recover() is called
- Not mentioned in the specs, but confirmed by Sun

The User Perspective

Choosing the Right Server Configuration

When Does a Server Crash?

- Overload is a common cause
  - Too many active requests
- Overload implies:
  - A maximum of active transactions
  - A maximum of prepared transactions
- Consequence for integration projects:
  - *A maximum risk of data inconsistency*

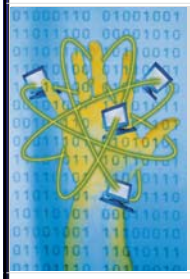
Recoverability Checklist

- Connector vendors:
  - is XA supported?
  - does isSameRM() work correctly?
  - is recover() implemented?
  - what XID format is allowed?
  - is multiplexing implemented correctly?
- Server configuration:
  - Does the JTA implementation support logging?
  - Is the server configured with XA connectors?

JBoss and JTA/XA

- By default, no logging (yet) in the JBoss JTA
  - OK for 95% of projects that don't need recoverability
- But: commercial products integrate with JBoss to do this
  - For the 5% niche market where this matters

Conclusion



- Recoverability is essential if:
  - more than one enterprise system is involved, and
  - the risk of data corruption is too high
- Recoverability requires:
  - A JTA implementation that supports logging
    - Available for JBoss as commercial modules
  - Connectors that fully support the XA protocol
    - Currently not yet common in open-source
    - A number of common pitfalls have been hinted at in this talk
- Servers tend to crash under overload
  - When the number of transactions is at a peak!!!