7 Fault Tolerant Distributed Transactions

Commit protocols

7.1 Subtransactions and distribution
7.2 Fault tolerance and commit processing
7.3 Requirements
7.4 One phase commit
7.5 Two phase commit

* based on Weikum / Vossen; Valduriez / Öszu; Garcia-Molina ; Reuter/ Gray
7.1 Transactions and Subtransactions

Transaction may be **nested** as opposed to **flat**.

Different semantic model of nested transactions:

- closed vs open
Closed Nested Transactions

Let $T$ be a parent transaction, $C_i$ child TA, $C_{ij}$ child TA of $C_i$, recursive.

**Commit rule:** $C_i$, $C_{ij}$... will be finally committed if all ancestors including $T$ commits.

**Abort rule:** If some $C_i$ aborts, all childs abort.
   Caveat: parent does not need to abort, if child aborts.

**Visibility rule:** if $C_i$ commits (locally!) data are visible to parent, but not to siblings.

**TA outcome basically controlled by $T$**
Open nested Transactions

Closed TA model **too restrictive** – compare federations of autonomous systems

**Open nested** transactions:

Subtransactions may commit independently
... and release resources.
⇒ needed: different undo mechanism

**Compensation** TA for undoing effects (if possible),
**Forward recovery** using savepoint.

More flexibility, less integrity
7.2 Fault tolerance and transaction

Primary problems of TA related to reliability:

Atomicity, Durability

Well known solution in centralized DBS

Save state information in a safe place
State information to be saved depends on
- failure model
- system aspects (e.g. buffer management)

Before image / after image / WAL is safe
Allows to reconstruct state of
- committed TAs, effects not yet stored in DB
- aborted TAs, effects partially in DB
- running TAs, effects partially in DB -> abort
Architectural model (centralized)

System model

Components of TA control

Basic principles for commit processing:
- write ahead log
- commit rule
Failures in distributed system

Partial failure makes it hard!!

"withdraw x from account a"

Has x already been added to b when S2 collapsed?

Avoid both: “add twice” and “lost add”

('exactly once' semantics)
Failures

Did S1 commit its subtransaction?  
i.e. did it receive the "commit" by the TA coordinator before the net / or S1 ? Collapsed?

Wanted:
Partial execution of one logical operation at different sites!
Types of failures

Transaction failures
Transaction aborts (unilaterally or due to deadlock)
Avg. 3% of transactions abort abnormally

System (site) failures
Failure of processor, main memory, power supply, …
Main memory contents are lost, but secondary storage contents are safe
Partial vs. total failure

Communication failures
Lost / undeliverable messages
Network partitioning
Failure Model

More failure types
Multiple failures
malevolent failures
Detectable failures

Failure Model
Fail-stop nodes

Network:
in-order msg., no spontaneous msg,
timeout, net partitions may occur
no persistent msg,
msg delivered eventually (makes life easier)
Distributed Commit

Commit coordinator

Transaction T

Action: a1,a2

Action: a3

Action: a4,a5

How to guarantee "all or nothing"?

Decision on "commit" and "abort" must be unanimous
Distributed Commit

"No-failure" mode

- Wait for "ack" of all actions (nodes)
- send "commit" to all participating nodes

Next to trivial – like many algorithms without resilience.

Participants (Ressource Managers) states:
Distributed Commit - Issues

Problems

– Transaction operates on multiple servers (resource managers)

– Distributed system may **fail partially** (server crashes, network failures) and create the potential danger of inconsistent decisions

– Global commit needs **unanimous agreement of all participants** (agents)

**Atomic commit problem**: find a protocol which ensures a unanimous decision also in case of failures.
7.2 Requirements for Atomic Commit

AC1 All participants finally come to the same decision (Uniform-agreement)

AC2 "Commit" decision can only be reached if all local decisions were "Commit" (Uniform validity)

AC3 A participant cannot reverse decision after deciding (Stability)

AC4 If there is no failure and all local decisions where "commit" then the overall decision is "commit" (Non-triviality)

AC5 All correct participants reach a decision (Non-blocking).
In all distributed systems:

**Safety conditions**: "nothing bad happens"

**Liveness Conditions**: "something happens"

**AC1 – AC3**: Safety
- unanimous, stable

**AC4, AC5**: Liveness
- Trivial solution of the AC would be: all participants always abort (AC4)
- Something will happen (AC5)

**AC1-AC5**: Non-blocking Atomic Commit (NB-AC) problem
Blocking

What does blocking mean?

A **blocking protocol** does - in case of failure - prevent the others from taking the final decision on the fate of the transaction.

Bad situation since resources of all participants blocked until recovery from failure.
In an asynchronous* distributed system, **there is no protocol which solves NB-AC.**

Idea of proof:

No way to decide between C and A without information about the fate of the TA. (**No independent recovery**)

* means: msg delay and process speed unbound
Relaxation of AC requirements

AC 4: too strong

"No failure \(\Rightarrow\) all decide commit"

AC 4':

"No participant suspected to fail \(\Rightarrow\) every participant reaches a commit decision."

(Non-Blocking weak atomic commit – NB-WAC)

"Suspected to fail": means there are failure detectors, e.g. timeouts, which detect crashes / communication failures, but may be wrong.

NB-WAC-Protocol based on a consensus protocol \(\rightarrow\) Paxos (see below)
7.4 One phase commit

Example: Calendar application
Application protocol: agreement on the date / time of some event.

\[ \text{e.g:} \]
".. everyone happy with suggested date? \]
  \text{if one participant votes no,} \\
  \text{coordinator makes new suggestion} \\
  \text{else commit (1-phase)}"

Agreement between nodes in processing phase, not during commit.
1PC: participant protocol

Every update is acknowledged, participant gives up veto right for the whole TA ⇒ one commit phase
Notation

Finite state automaton
different for
  - participants
  - coordinator
State transition labeled by
  msg received / msg send

transition fct $\delta$: inputs $X$ states $\rightarrow$ states
output fct $\lambda$: inputs $X$ states $\rightarrow$ output

Any statechart type is ok
Characteristics of 1PC

Blocking?
Yes! When?
Two types of blocking:
- participant failure
- coordinator failure – more serious, why?

Window of uncertainty in failure free case?

Number of messages for commit /abort?
Suppose $n$ participants.
More involved task

$n$ participants, each having a variable $x_i$

- clients send increments ("+j") to each of them
- no individual ack of an increment operation, (but of msg received)

--- end of operation phase ------

Condition for successful operation: **all** increments successful (no overflow, or alike)

If not successful: participants reset $x_i$

Commit coordinator has to decide!

Commit phase? **1PC is not sufficient to come to a unanimous result!** Why?

--- work phase

--- commit phase