



Übung 3: Verteilte Datenhaltung

1. Serialisierung

Konstruieren Sie Historien aus drei Transaktionen T1, T2 und T3, die folgende Merkmale aufweisen:

1. Die serielle Reihenfolge ist T1 vor T2 vor T3.
2. T3 beendet seine Arbeit vor T2; T2 wiederum vor T1.

Zudem sollen folgende Bedingungen gelten (betrachten Sie jede einzeln):

- a. Recoverable
- b. Cascadeless
- c. Strict
- d. Seriell
- e. COCSR

Falls es in einem der genannten Fälle (a bis e) nicht möglich ist, eine Historie zu konstruieren, erklären Sie, warum dies so ist.

1. Serialisierung

- a. $w_1[x] w_2[x] w_3[x] c_3 c_2 c_1$ ist recoverable
(Recoverability ist nur in Gegenwart von read-Operationen definiert)
- b. $w_1[x] w_2[x] w_3[x] c_3 c_2 c_1$ ist cascadeless
(Cascadelessness ist nur in Gegenwart von read-Operationen definiert)

1. Serialisierung

- c. Strictness erfordert Cascadelessness $w_1[x] < r_2[x] \rightarrow c_1 < r_2[x]$
S. impliziert zusätzlich $w_1[x] < w_2[x] \rightarrow c_1 < w_2[x]$
Eine strikte Historie mit Merkmalen 1 und 2 ist nicht möglich
- d. Seriality: Eine Transaktion endet (committed) bevor die nächste beginnt
(Serielle Reihenfolge = commit order)
Eine serielle Historie mit Merkmalen 1 und 2 ist nicht möglich
- e. COCSR erfordert lokale Strictness.
Eine COCSR Historie mit Merkmalen 1 und 2 ist nicht möglich



2. Timeout in 2PC

Nehmen Sie an, in einem linearen, nach 2PC arbeitenden verteilten System sei die maximale Laufzeit einer Nachricht von einem Knoten zu seinem rechten oder linken Nachbarknoten konstant.

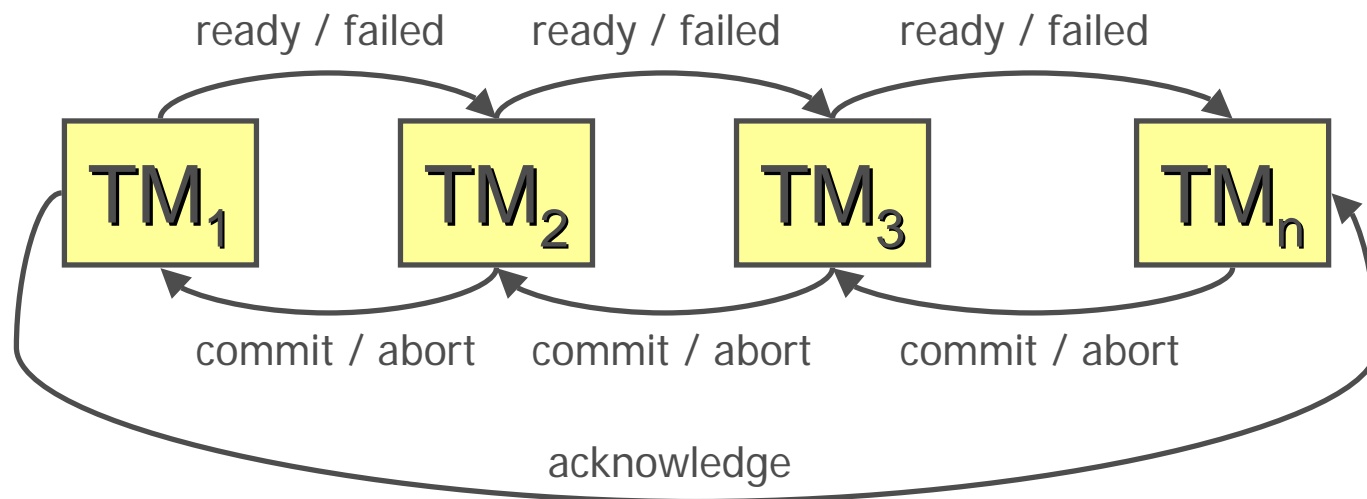
Wie sollte der Timeout gewählt werden, in Abhängigkeit von dieser maximalen Laufzeit, der Position des jeweiligen Knotens in der linearen Struktur und des Nachrichtentyps?

2. Timeout in 2PC

T_{delay} : Maximale Laufzeit einer Nachricht von einem Knoten zu seinem rechten oder linken Nachbarn

Sei k die Position eines Knoten s , gezählt von TM_1

Sei n die Anzahl der beteiligten Knoten



2. Timeout in 2PC

in 2PC

$T_{\text{out}}(\text{agent}) \geq \text{Zeit bis ready / failed} + \text{Zeit bis commit / abort}$

in linear 2PC:

$T_{\text{out}}(\text{agent}) \geq \text{Zeit bis ready / failed} + \text{Zeit bis commit / abort}$
 $\geq (2n-2k) * T_{\text{delay}} + (k-1)*T_{\text{delay}}$



3. Unsicherheit in 2PC

- a. Was ist die Phase der Unsicherheit in 2PC?
- b. Gibt es in linearem 2PC einen Knoten, der sich niemals in dieser Phase befindet? Wenn ja, welche(r)? Wenn nicht, beschreiben Sie die Unsicherheitsphase für jeden Knoten.



3. Unsicherheit in 2PC

- a. Phase der Unsicherheit:
Phase zwischen dem Vote eines Knotens und der Benachrichtigung über das Abstimmungsergebnis vom Master-Knoten
- b. Knoten ohne Phase der Unsicherheit:
Knoten TM_n kenne das Abstimmungsergebnis unmittelbar nach seinem eigenen Vote, da er der letzte in der Reihe ist. Daher gibt es für ihn keine Phase der Unsicherheit.

4. Three Phase Commit (3PC)

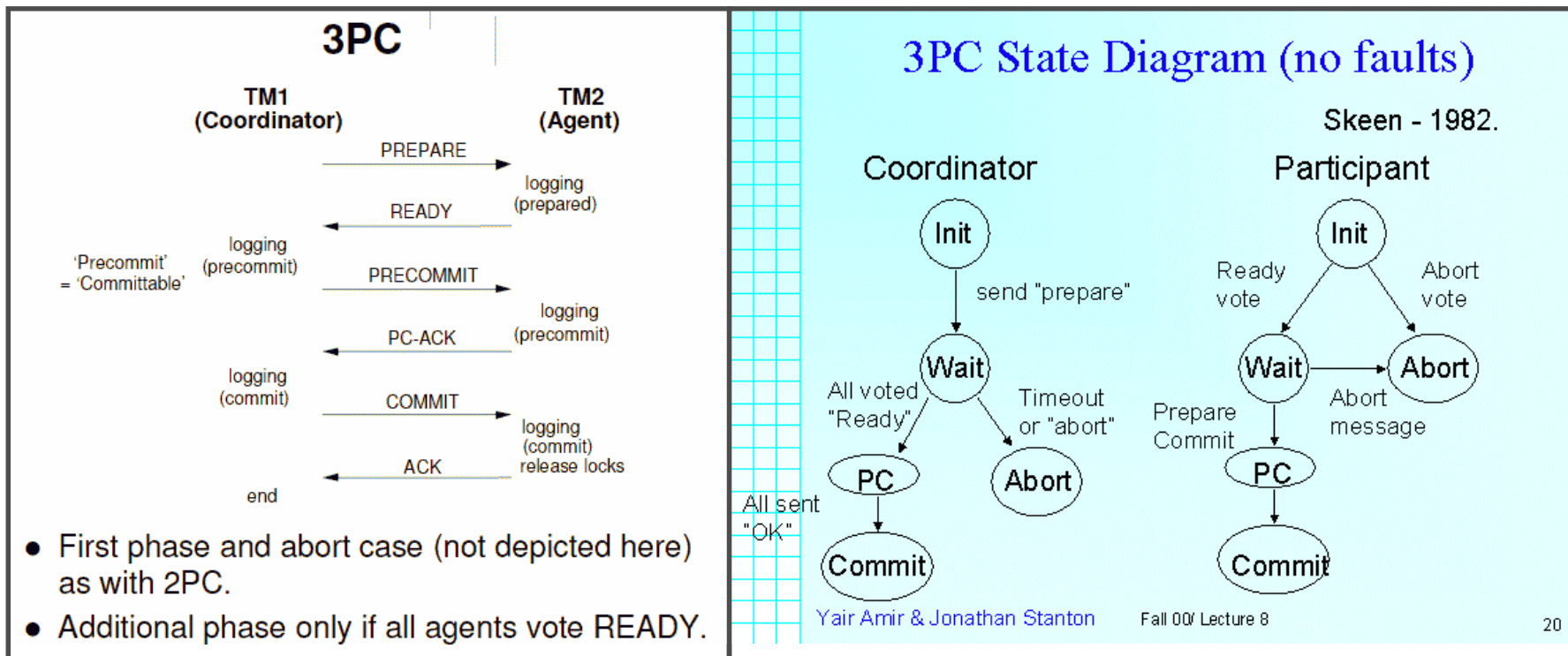
Beschreiben Sie die Variante von 3PC, in der konsistente Entscheidungen garantiert werden können, sofern nur seitens der Knoten Fehler auftreten (site failures).

Blocking: Wenn der letzte COMMIT gesendet wurde, haben die einzelnen Knoten unterschiedliche Zustände in ihren lokalen Datenbanken. Auf manchen sind die Änderungen bereits sichtbar, auf anderen nicht, da sie ihren Commit noch nicht ausgeführt haben.

Idee: Einschlebung eines PRECOMMIT, das es dem Koordinator-Knoten ermöglicht, diese Phase zu vermeiden

PRECOMMIT: Ende der Phase der Unsicherheit, aber noch kein COMMIT. Wenn der Koordinator zusammenbricht, entscheidet jeder Knoten selbst über sein weiteres Vorgehen, anstatt zu blockieren.

4. Three Phase Commit (3PC)





5. Presumed Commit

- a. Beschreiben Sie kurz, was ein Presumed Abort ist.
- b. Beschreiben Sie das Presumed Commit Protokoll. Ist dieses Protokoll symmetrisch zum Presumed Abort? Erläutern Sie Ihre Antwort.

5. Presumed Commit

Atomare Commit-Protokolle:

- 2PC: Kann blockieren
- 3PC: Zwei Arten:
 1. Toleriert Site Failures → kein Blockieren außer bei komplettem Versagen des gesamten Systems (kann zu Inkonsistenzen führen)
 2. Toleriert Site Failures und Communication Failures → Kann blockieren

Presumed Abort

- Objective: less messages and less log entries.
- If after failure coordinator log file does not contain commit log entry: decide **abort**.
- *Presumed abort* – incorporated in several products and in standards (ISO/OSI TP, X/Open DTP).
- Advantages:
 - ◆ Coordinator does not need to write abort log entry synchronously.
 - ◆ ACK messages for failed transactions superfluous,
 - ◆ same with end log entries at coordinator and intermediate nodes.
- However, no savings with successful global transactions.
- Particularly effective: further optimization for subtransactions that only read (see below).

6. Logging und Recovery in 2PC / linearem 2PC / 3PC

- a. Warum werden in Datenbank-Systemen UNDO- und REDO-Informationen geloggt?
- b. Welche Arten von Logging kommen in 2PC, linearem 2PC und 3PC zur Anwendung? Unterscheiden Sie in Ihrer Antwort zwischen Koordinator und anderen Knoten, und begründen Sie jede Log-Operation.
- c. Nehmen Sie an, der Platz auf der Festplatte sei knapp. Welche Log-Einträge können nach den einzelnen Phasen wieder gelöscht werden?

6. Logging und Recovery in 2PC / linearem 2PC / 3PC

Zweck von UNDO: Entfernen / Rückgängig-Machen von Einträgen, die zu einer aborteten Transaktion gehören, aber bereits in RAM / auf Festplatte geschrieben wurden, bevor es zum Abort kam

Zweck von REDO: Finales Ausführen (Schreiben in RAM / auf Festplatte) von Operationen, die zu einer commiteten Transaktion gehören, aber noch nicht geschrieben wurden

- Was bedeutet Kaskadierung in diesem Kontext?
- Warum Logging statt direktem Schreiben der Daten?

6. Logging / Recovery (2PC)

Koordinator-Knoten:

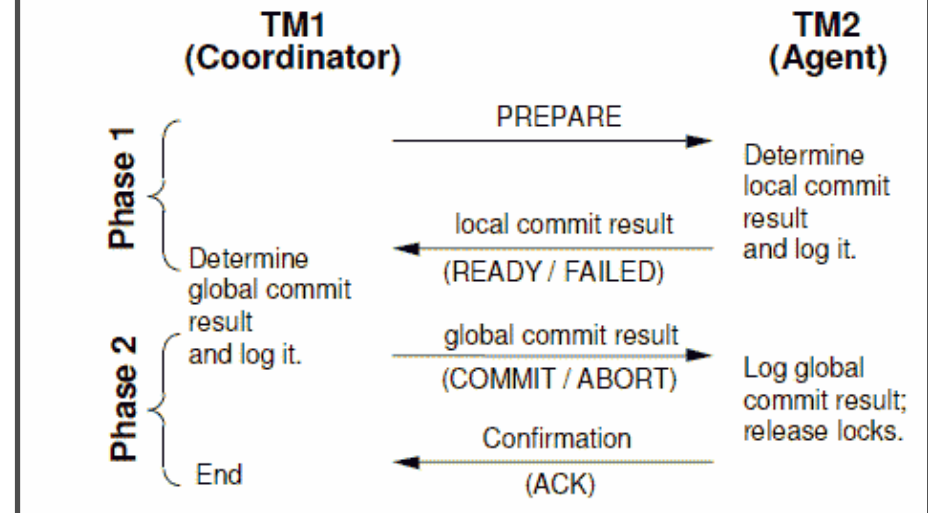
Erhalte Votes

Logge Ende (nun
können alle zuvor
gemachten Log-
Einträge gelöscht
werden)

Andere Knoten:

PREPARED-Status,
finde lokalen Vote
(READY / FAILED)
Warte auf globalen
ABORT oder COMMIT

- Two-phase commit (2PC),
- centralized communication structure.
- **Flow of messages:**



6. Logging / Recovery (L2PC)

PREPARED-Status, finde
lokalen Vote (READY /
FAILED)

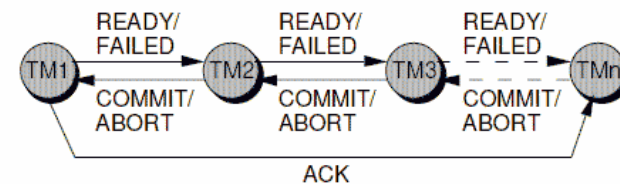
Warte auf globalen ABORT
oder COMMIT

Logge Ende

Votes brauchen nicht geloggt
zu werden, da mit
eigenem Vote korreliert →
ein Log-Eintrag gespart

Linear Two-Phase Commit (1)

- Commit processing sequentially via the TMs of the n nodes participating in the global TA.
- Phase 1: communication 'in the forward direction' from coordinator (TM1) to last agent (TM n);
Phase 2: other direction.



6. Logging Recovery (3PC)

Precommit muß geloggt werden, damit bei Ausfall des Koordinators ein neuer bestimmt werden kann

Koordinator-Knoten:

Sammele Votes

Sende PRECOMMIT

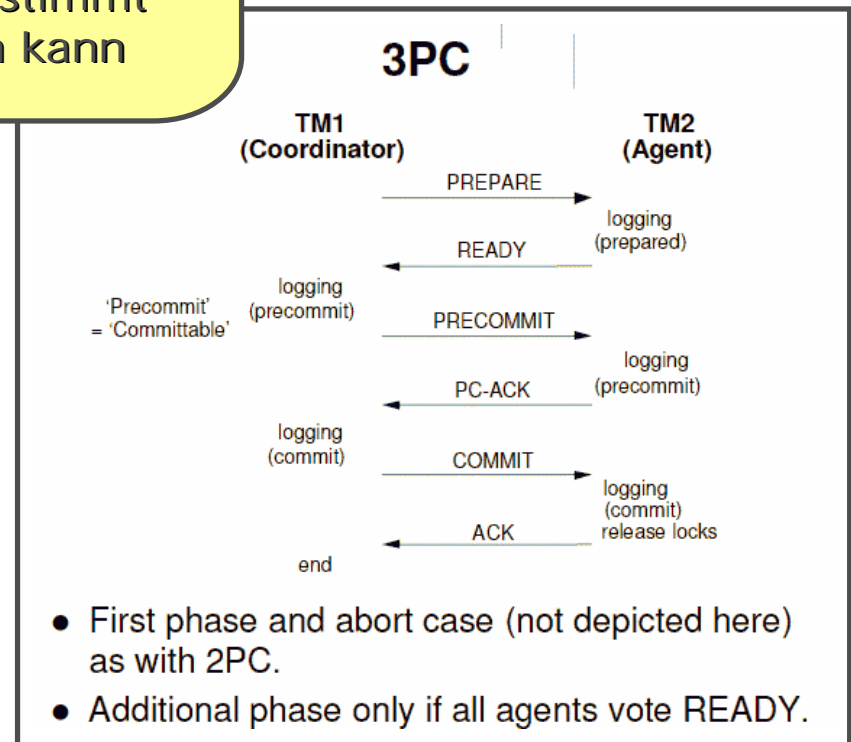
Logge Ende (nun können alle zuvor gemacht Log-Einträge gelöscht werden)

Andere Knoten:

PREPARED-Status, finde lokalen Vote (READY / FAILED)

Erwarte PRECOMMIT

Warte auf globalen ABORT oder COMMIT



7. Read-Only Sub-Transaktionen

- a. Welche Beziehung besteht zwischen Non-Repeatable / Repeatable Reads und Atomic Commitment?
- b. Beschreiben Sie Eigenschaften, die eine Transaktion erfüllen muß, damit Sie auch bei Auftreten von Non-Repeatable Reads mit hoher Wahrscheinlichkeit korrekt abläuft.
- c. Nehmen Sie an, eine solche Transaktion soll nun auf einem verteilten Datenbank-System ausgeführt werden. Welche Schritte des Protokolls können entfallen, wenn das Isolations-Level gesenkt wird?

7. Read-Only Sub-Transaktionen

- a. Welche Beziehung besteht zwischen Non-Repeatable / Repeatable Reads und Atomic Commitment?

AC1: All processes that reach a decision reach the same one.

AC2: A process cannot reverse its decision after it has reached one.

AC3: The Commit decision can only be reached if *all* processes voted Yes.

AC4: If there are no failures and all processes voted Yes, then the decision will be to Commit.

AC5: Consider any execution containing only failures that the algorithm is designed to tolerate. At any point in this execution, if all existing failures are repaired and no new failures occur for sufficiently long, then all processes will eventually reach a decision.

7. Read-Only Sub-Transaktionen

Warum Commit auch bei Read Only?

Mehrfaches Lesen desselben Wertes liefert unterschiedliche Ergebnisse, da der Wert zwischenzeitlich von einer anderen Transaktion geändert worden sein kann.

Atomic Commitment (generic term für 2PC, 3PC etc.):
Beugt Non-Repeatable Reads vor, Isolations-Levels:

- | | |
|-----------------------------|---|
| 0 – <i>read uncommitted</i> | (Dirty und Non-Repeatable Read, Phantome) |
| 1 – <i>read committed</i> | (Non-Repeatable Read, Phantome) |
| 2 – <i>repeatable read</i> | (Phantome) |
| 3 – <i>serializable</i> | (keine Probleme, aber langsam) |



7. Read-Only Sub-Transaktionen

b. Beschreiben Sie Eigenschaften, die eine Transaktion erfüllen muß, damit Sie auch bei Auftreten von Non-Repeatable Reads mit hoher Wahrscheinlichkeit korrekt abläuft.

... Innerhalb einer Transaktion jeden Wert nur einmal lesen ... 😊

7. Read-Only Sub-Transaktionen

c. Nehmen Sie an, eine solche Transaktion soll nun auf einem verteilten Datenbank-System ausgeführt werden. Welche Schritte des Protokolls können entfallen, wenn das Isolations-Level gesenkt wird?

... Teilnahme an COMMIT-Protokollen kann entfallen → Schnellere Ausführung durch bessere Parallelisierung, Locking wird leichtgewichtig und weniger restriktiv



8. One Phase Commit (1PC)

Beschreiben sie kurz den One Phase Commit. Welche Voraussetzungen müssen erfüllt sein, wo liegen die Vor- und Nachteile?

Anforderungen:

- Kurze Sub-Transaktionen

- Nur eine Operation auf einer externen Datenbasis

Idee: Kombiniere PREPARE-Nachricht mit letzter Operations-Nachricht: DONE-Nachricht der Operation liefert gleichzeitig READY- oder FAILED-Nachricht der PREPARE-Phase



8. One Phase Commit (1PC)

Warum nicht immer 1PC?

Problem: Phase der Unsicherheit bei Transaktionen unterschiedlicher Laufzeit sehr lang

Problem bei Operationen mit mehr als einer externen Datenbank: Eine könnte COMMIT (READY / FAILED) senden, die andere könnte nachträgliche Änderungen erfordern

→ ABORT, obwohl READY schon gesendet.

9. Total Failures

Nehmen Sie den Total-Ausfall eines verteilten Datenbank-Systems an. Wie funktioniert die Recovery in 2PC / 3PC? Welche Knoten und welche Informationen müssen zur Verfügung stehen, bevor das System mit der Recovery beginnen kann?

2PC: Warte auf Koordinator, da er den globalen Status der Transaktion und die Votes der anderen Knoten kennt

3PC: Warte auf den letzten Knoten, da er ein PRECOMMIT in seinem Log hat.

10. Replication vs. Fragmentation

- a. Beschreiben Sie die Vor- und Nachteile von Replikation im Vergleich zu Fragmentierung. Betrachten Sie Verfügbarkeit, Performanz von Lese- und Schreib-Operationen, Ausfallsicherheit, Verteilung von Schema-Informationen, etc.
- b. In welchen Systemen sind verteilte Transaktionen aufwändiger, in replizierten oder in fragmentierten Systemen?
- c. Definieren und unterscheiden Sie Lazy Replication und Eager Replication.

10. Replication vs. Fragmentation

- a. Beschreiben Sie die Vor- und Nachteile von Replikation im Vergleich zu Fragmentierung. Betrachten Sie Verfügbarkeit, Performanz von Lese- und Schreib-Operationen, Ausfallsicherheit, Verteilung von Schema-Informationen, etc.

Replikation

Erhöhte Verfügbarkeit

Erhöhte Performanz von Read-Operationen
(mehrere parallel auf mehreren Knoten)

Performanz von Write-Operationen bleibt gleich

Komplexes Transaktions-Management

Fragmentierung

10. Replication vs. Fragmentation

- b. In welchen Systemen sind verteilte Transaktionen aufwändiger, in replizierten oder in fragmentierten Systemen?

Transaktionen in replizierten Systemen sind aufwändiger, da in einem fragmentierten System ein Knoten den exklusiven Zugriff auf einen Datensatz hat.

- c. Definieren und unterscheiden Sie Lazy Replication und Eager Replication.

lazy: Transaktion zum Update der Replikate ist getrennt von der Transaktion, die die eigentliche Änderung durchführt (asynchron)

eager: Die Replikate werden sofort und in derselben Transaktion aktualisiert, die die eigentliche Änderung durchführt (synchron)

Fragen ??
(vor allem zum bisherigen
Stoff der Vorlesung)

Fragen jederzeit an:
sautter@ipd.uka.de