

Database System Internals

Two-Phase Commit (2PC)

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

March 11, 2020 CSE 444 - Winter 2020 1

1

Course Evaluations

Please submit evaluation feedback!

<https://uw.iasystem.org/survey/219322>

March 11, 2020 CSE 444 - Winter 2020 2

2

References

- Ullman book: Section 20.5
- Ramakrishnan book: Chapter 22

March 11, 2020 CSE 444 - Winter 2020 5

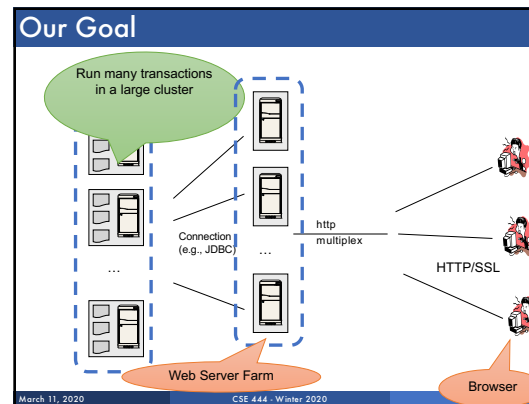
5

We are Learning about **Scaling DBMSs**

- **Scaling the execution of a query**
 - Parallel DBMS
 - MapReduce
 - Spark
- **Scaling transactions**
 - Distributed transactions
 - Replication
 - Scaling with NoSQL and NewSQL

March 11, 2020 CSE 444 - Winter 2020 6

6



7

Transaction Scaling Challenges

- **Distribution**
 - There is a limit on transactions/sec on one server
 - Need to partition the database across multiple servers
 - If a transaction touches one machine, life is good!
 - If a transaction touches multiple machines, ACID becomes extremely expensive! Need two-phase commit
- **Replication**
 - Replication can help to increase throughput and lower latency
 - Create multiple copies of each database partition
 - Spread queries across these replicas
 - Easy for reads but writes, once again, become expensive!

March 11, 2020 CSE 444 - Winter 2020 8

8

Distributed Transactions

- **Concurrency control**
- **Failure recovery**
 - Transaction must be committed at all sites or at none of the sites!
 - No matter what failures occur and when they occur
 - Two-phase commit protocol (2PC)

March 11, 2020 CSE 444 - Winter 2020 9

9

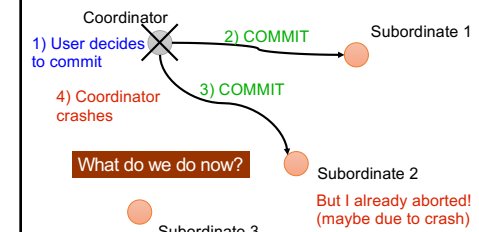
Distributed Concurrency Control

- In theory, different techniques are possible
 - Pessimistic, optimistic, locking, timestamps
- In practice, distributed two-phase locking
 - Simultaneously hold locks at all sites involved
- **Deadlock detection techniques**
 - Global wait-for graph (not very practical)
 - Timeouts
- If deadlock: **abort least costly local transaction**

March 11, 2020 CSE 444 - Winter 2020 10

10

Two-Phase Commit: Motivation



March 11, 2020 CSE 444 - Winter 2020 11

11

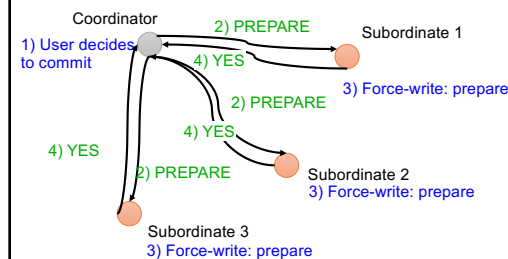
Two-Phase Commit Protocol

- One coordinator and many subordinates
 - Phase 1: prepare
 - All subordinates must flush tail of write-ahead log to disk before ack
 - Must ensure that if coordinator decides to commit, they can commit
 - Phase 2: commit or abort
 - Log records for 2PC include transaction and coordinator ids
 - Coordinator also logs ids of all subordinates
- **Principle**
 - Whenever a process makes a decision: vote yes/no or commit/abort
 - Or whenever a subordinate wants to respond to a message: ack
 - First **force-write a log record** (to make sure it survives a failure)
 - Only then send message about decision
- **"Forget"** completed transactions at the very end
 - Once synchronized on whether the transaction has committed or aborted, all nodes can stop logging any more information about that transaction

March 11, 2020 CSE 444 - Winter 2020 12

12

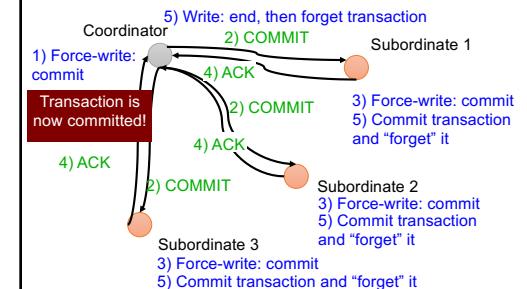
2PC: Phase 1, Prepare



March 11, 2020 CSE 444 - Winter 2020 13

13

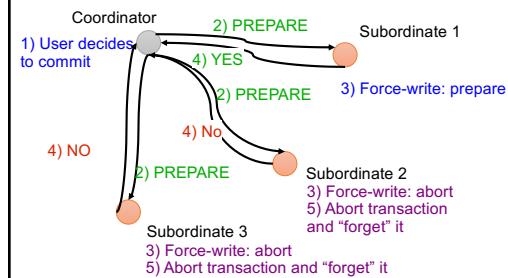
2PC: Phase 2, Commit



March 11, 2020 CSE 444 - Winter 2020 14

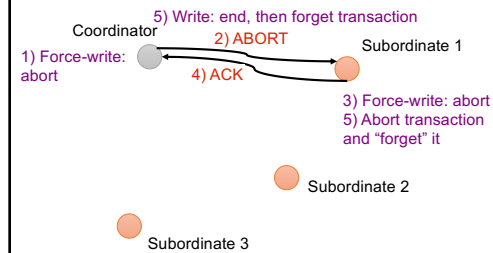
14

2PC with Abort – Phase 1



15

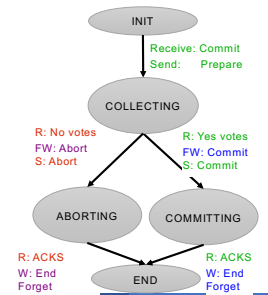
2PC with Abort – Phase 2



16

Coordinator State Machine

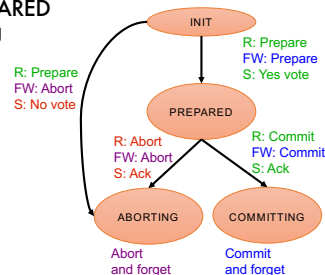
- All states involve waiting for messages



17

Subordinate State Machine

- INIT and PREPARED involve waiting



18

Handling Site Failures

- Approach 1: no site failure detection
 - Can only do retrying & blocking
- Approach 2: timeouts
 - Since unilateral abort is ok,
 - Subordinate can timeout in init state
 - Coordinator can timeout in collecting state
 - Prepared state is still blocking
- 2PC is a blocking protocol

19

Site Failure Handling Principles

- Retry mechanism
 - In prepared state, periodically query coordinator
 - In committing/aborting state, periodically resend messages to subordinates
- If doesn't know anything about transaction respond "abort" to inquiry messages about fate of transaction
- If there are no log records for a transaction after a crash then abort transaction and "forget" it

20

Observations

- Coordinator keeps transaction in transactions table until it receives all acks
 - To ensure subordinates know to commit or abort
 - So acks enable coordinator to "forget" about transaction
- After crash, if recovery process finds no log records for a transaction, the transaction is presumed to have aborted
- Read-only subtransactions: no changes ever need to be undone nor redone

March 11, 2020 CSE 444 - Winter 2020 22

22

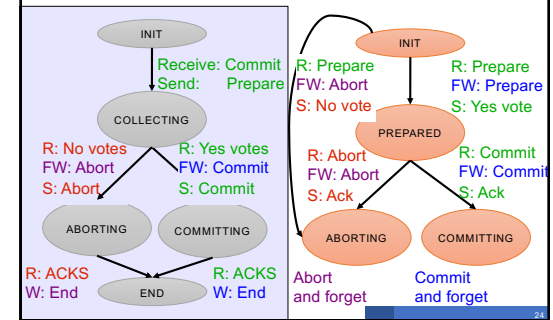
Presumed Abort Protocol

- Optimization goals
 - Fewer messages and fewer force-writes
- Principle
 - If nothing known about a transaction, assume ABORT
- Aborting transactions need no force-writing
- Avoid log records for read-only transactions
 - Reply with a READ vote instead of YES vote
- Optimizes read-only transactions

March 11, 2020 CSE 444 - Winter 2020 23

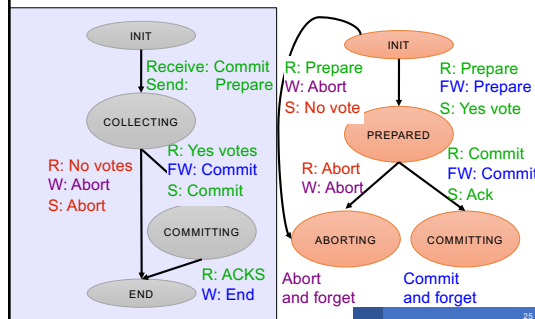
23

2PC State Machines (repeat)



24

Presumed Abort State Machines



March 11, 2020 CSE 444 - Winter 2020 25

25

Presumed Abort Protocol

- With this protocol, we have cut an entire state from the coordinator state machine
- Less waiting and log writes

These are the basics of 2-PC!

March 11, 2020 CSE 444 - Winter 2020 26

26

Course Evaluations

Please submit evaluation feedback!

<https://uw.iasystem.org/survey/219322>

March 11, 2020 CSE 444 - Winter 2020 27

27