CLOUD COMPUTING

Distributed Transactions

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

Transactions

- **Transaction** is an operation composed of a number of discrete steps
- All the steps must be completed for the transaction to be **committed**. The results are made permanent else the transaction is **aborted** and the state of the system **reverts** to what it was before the transaction started E.g. Buying a house
- Basic Operations
  - Transaction primitives:
    - Begin transaction: mark the start of a transaction
    - End transaction: mark the end of a transaction; try to commit
    - Abort transaction: kill the transaction, restore old values
    - Read/write data from files (or object stores): data will have to be restored if the transaction is aborted.

Transactions : Properties of Transactions (ACID)

ACID :

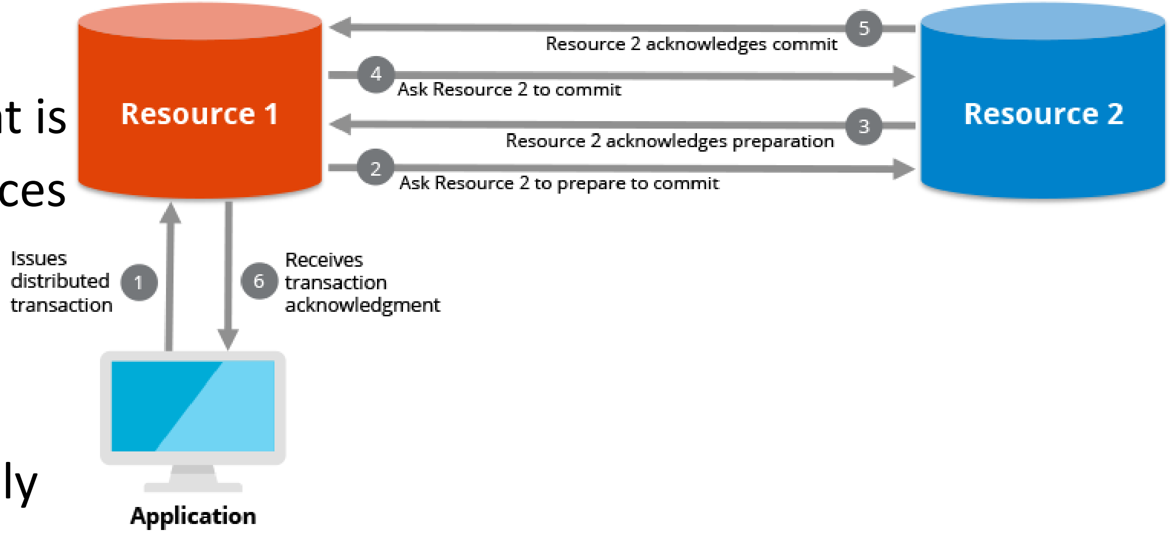
- Atomic
  - The transaction happens as a single indivisible action. Others do not see intermediate results. All or nothing.
- Consistent
  - If the system has invariants, they must hold after the transaction. E.g., total amount of money in all accounts must be the same before and after a “transfer funds” transaction.
- Isolated (Serializable)
  - If transactions run at the same time, the final result must be the same as if they executed in some serial order.
- Durable
  - Once a transaction commits, the results are made permanent. No failures after a commit will cause the results to revert.

Transactions and different types of transactions :

Nested Transactions

- Nested Transaction is a top-level transaction which may create sub-transactions
- Problem:
  - Sub-transactions may commit (results are durable) but the parent transaction may abort.
- One solution : private workspace
  - Each sub-transaction is given a private copy of every object it manipulates. On commit, the private copy displaces the parent's copy (which may also be a private copy of the parent's parent)

Distributed Transactions


Resource 1
Resource 2
5 Resource 2 acknowledges commit
4 Ask Resource 2 to commit
3 Resource 2 acknowledges preparation
2 Ask Resource 2 to prepare to commit
Issues distributed transaction 1
6 Receives transaction acknowledgment
Application

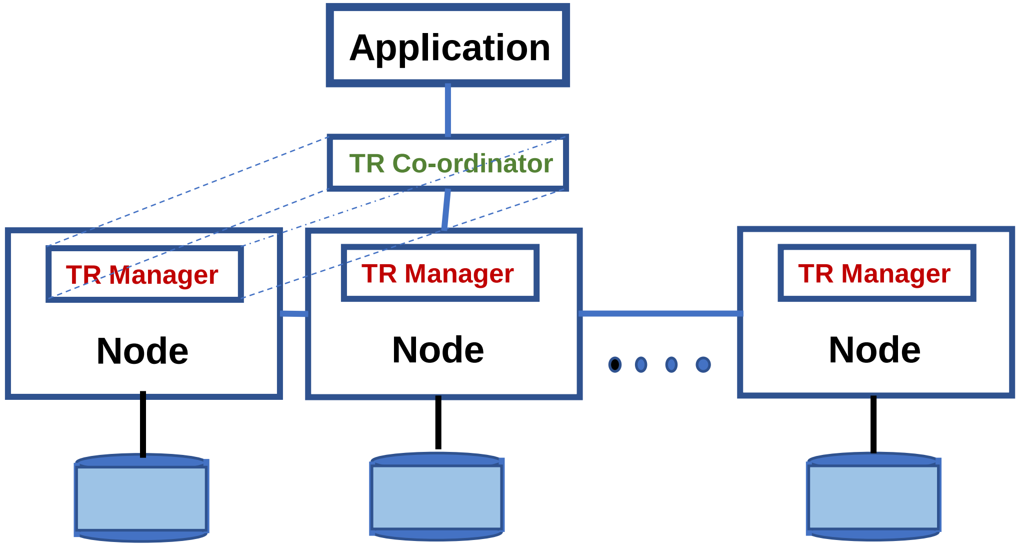
- A distributed transaction is a set of operations on data that is performed across two or more data repositories or resources across different systems
- Challenge: handle machine, software, & network failures while preserving transaction integrity.
- There are two possible outcomes all operations successfully complete, or none of the operations are performed at all due to a failure somewhere in the system
- In the second outcome if some work was completed prior to the failure, that work will be reversed to ensure no net work was done. This type of operation is in compliance with the "ACID" principles to ensure data integrity

Distributed Transaction System Architecture

- A system architecture supporting distributed transactions, has multiple data repositories hosted on different nodes connected by a network.
- Transaction may access data at several nodes/sites.
- Each site has a

local transaction manager responsible for:

- Maintaining a log for recovery purposes
- Participating in coordinating the concurrent execution of the transactions executing at that site.
- Responsible for sub-transactions on that system
- Performs prepare, commit and abort calls for sub-transactions
- Each sub-transaction must agree to commit changes before the transaction can complete



Transaction coordinator coordinating activities across the data repositories

- Periodically a local transaction manager is nominated as a local coordinator
- Starting the execution of transactions that originate at the site.
- Distributing sub transactions at appropriate sites for execution.
- Coordinating the termination of each transaction that originates at the site, which may result in the transaction being committed at all sites or aborted at all sites.

Concurrency Control

- All concurrency mechanisms must preserve data consistency and complete each atomic action in finite time
- Important capabilities are
  a) Be resilient to site and communication link failures.

  b) Allow parallelism to enhance performance requirements.

  c) Incur optimal cost and optimize communication delays

  d) Place constraints on atomic action.

Commit Protocols

- Commit protocols are used to ensure atomicity across sites
  - A transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.
  - Not acceptable to have a transaction committed at one site and aborted at another.

- The two-phase commit (2 PC) protocol is widely used.

Two Phase Commit Protocol - Phase 1

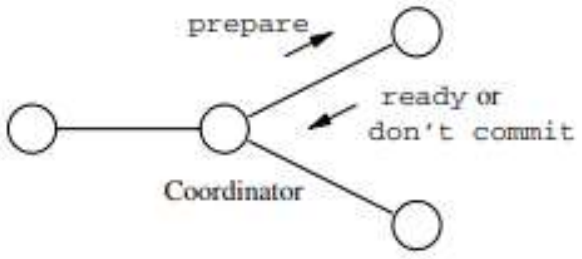
Lets consider a transaction is named as “**T**”

1. The coordinator places a log record **prepare T** on the log at its site.
2. The coordinator sends to each component’s site the message **prepare T**.
3. Each site receiving the message prepare its component of the transaction “T”.
4. If a site wants to commit its component, it must enter a state called **pre-committed** with a **Ready T** message. Once in the **pre-committed** state, the site cannot ***abort*** its component of T without a directive to do so from the coordinator

   So, after **prepare T** is received, Perform whatever steps necessary to be sure the local component of T will not have to abort

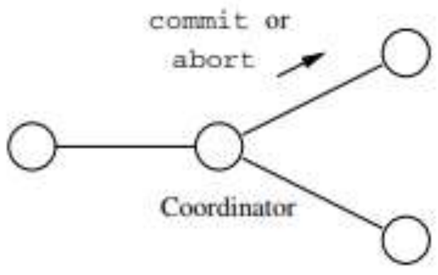
   If everything is fine and it ready to commit, then place the record **Ready T** on the local log and flush the log to disk and send **Ready T** message to the coordinator

   If the site wants to abort its component of T, then it logs the record **Don’t Commit T** and sends the message **Don’t commit T** to the coordinator



Two Phase Commit : Phase 1	
Co-Ordinator	**Related Nodes (Sites)**
▪ **Write prepare to commit T to *log* at its site** ▪ **Send *prepare to commit* message**	▪ **Receive *prepare* message** ▪ **Work on the components towards T** ▪ **If ready to commit get into pre-committed state for T and place *Ready T* in the local log and send *Ready T* message to the coordinator (holds locks..)** ▪ **If not ready to commit place *Don’t commit T* to the local log and send *Don’t commit T* message to the coordinator** ▪ **Wait for message from coordinator**
▪ **Wait for reply from each related node**	

Two Phase Commit Protocol - Phase 2

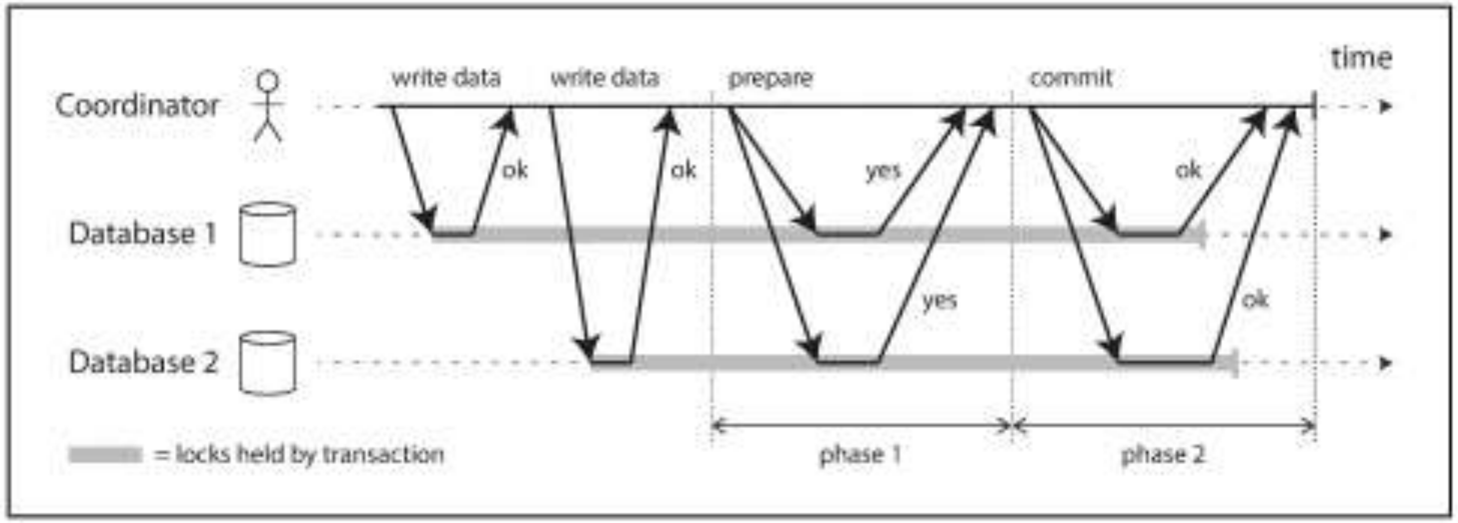


1. If the coordinator has received ready T from all components of T, then it decides to commit T. The coordinator logs **Commit T** at its site and then sends message ***commit T*** to all sites involved in T
2. If the coordinator has received don't commit T from one or more sites, it logs **Abort T** at its site and then sends ***abort T*** messages to all sites involved in T
3. If a site receives a ***commit T*** message, it commits the component of T at that site, releases the locks ..,logging **Commit T** as it does.
4. If a site receives the message ***abort T***, it aborts T, releases locks and writes the log record **Abort T**

Two Phase Commit : Phase 2	
Co-Ordinator	**Related Nodes (Sites)**
▪ If *Ready T* has been received from all nodes, then write *commit T* to the local log and Send *commit T* message ▪ If Don't commit T is received from any of the related nodes, then write *Abort T* to the local log and send *Abort T* is sent to all related nodes (sites) ▪ Wait for *Done* message and clear up all states	(If in the Pre-Committed State, continue to hold the locks) ▪ Receive *Commit T* or *Abort T* message ▪ If *Commit T* is received, commit the component of T at the site, release locks .. and place *Commit T* to local log and send *Done* message to the co-ordinator ▪ If *Abort T* is received, roll back all changes, release locks .. and place *Abort T* to the local log and send *Done* message to the Co-ordinator

Two Phase Commit Protocol

A successful execution of two-phase commit



THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu