Outline

- Introduction & architectural issues
- Data distribution
- Distributed query processing
- Distributed query optimization
- Distributed transactions & concurrency control
- Distributed reliability
  - Logging
  - Distributed commit protocols
- Data replication
- Parallel database systems
- Database integration & querying
- Peer-to-Peer data management
- Stream data management
- MapReduce-based distributed data management

Reliability

Problem:

How to maintain

atomicity

durability

properties of transactions
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

- **Media failures**
  - Failure of secondary storage devices such that the stored data is lost
  - Head crash/controller failure (?)

- **Communication failures**
  - Lost/undeliverable messages
  - Network partitioning

Local Recovery Management – Architecture

- **Volatile storage**
  - Consists of the main memory of the computer system (RAM).

- **Stable storage**
  - Resilient to failures and loses its contents only in the presence of media failures (e.g., head crashes on disks).
  - Implemented via a combination of hardware (non-volatile storage) and software (stable-write, stable-read, clean-up) components.
Recovery Information

Database Log
Every action of a transaction must not only perform the action, but must also write a log record to an append-only file.

Logging
The log contains information used by the recovery process to restore the consistency of a system. This information may include:

- transaction identifier
- type of operation (action)
- items accessed by the transaction to perform the action
- old value (state) of item (before image)
- new value (state) of item (after image)
  ...

Why Logging?

Upon recovery:
- all of $T_1$’s effects should be reflected in the database (REDO if necessary due to a failure)
- none of $T_2$’s effects should be reflected in the database (UNDO if necessary)

REDO Protocol

- REDO’ing an action means performing it again.
- The REDO operation uses the log information and performs the action that might have been done before, or not done due to failures.
- The REDO operation generates the new image.
UNDO Protocol

- UNDO’ing an action means to restore the object to its before image.
- The UNDO operation uses the log information and restores the old value of the object.

When to Write Log Records Into Stable Store

Assume a transaction $T$ updates a page $P$

- Fortunate case
  - System writes $P$ in stable database
  - System updates stable log for this update
  - SYSTEM FAILURE OCCURS!... (before $T$ commits)
  
  We can recover (undo) by restoring $P$ to its old state by using the log

- Unfortunate case
  - System writes $P$ in stable database
  - SYSTEM FAILURE OCCURS!... (before stable log is updated)

  We cannot recover from this failure because there is no log record to restore the old value.

- Solution: Write-Ahead Log (WAL) protocol
Write-Ahead Log Protocol

- Notice:
  - If a system crashes before a transaction is committed, then all the operations must be undone. Only need the before images (undo portion of the log).
  - Once a transaction is committed, some of its actions might have to be redone. Need the after images (redo portion of the log).

- WAL protocol:
  - Before a stable database is updated, the undo portion of the log should be written to the stable log.
  - When a transaction commits, the redo portion of the log must be written to stable log prior to the updating of the stable database.

Distributed Reliability Protocols

- Commit protocols
  - How to execute commit command for distributed transactions.
  - Issue: how to ensure atomicity and durability?

- Termination protocols
  - If a failure occurs, how can the remaining operational sites deal with it.
  - Non-blocking: the occurrence of failures should not force the sites to wait until the failure is repaired to terminate the transaction.

- Recovery protocols
  - When a failure occurs, how do the sites where the failure occurred deal with it.
  - Independent: a failed site can determine the outcome of a transaction without having to obtain remote information.

- Independent recovery ⇒ non-blocking termination
Two-Phase Commit (2PC)

**Phase 1**: The coordinator gets the participants ready to write the results into the database

**Phase 2**: Everybody writes the results into the database

- **Coordinator**: The process at the site where the transaction originates and which controls the execution
- **Participant**: The process at the other sites that participate in executing the transaction

**Global Commit Rule**:
- The coordinator aborts a transaction if and only if at least one participant votes to abort it.
- The coordinator commits a transaction if and only if all of the participants vote to commit it.

Centralized 2PC

![Centralized 2PC Diagram]
### 2PC Protocol Actions

**Coordinator**
- **INITIAL**
  - Write begin transaction in log
  - Write commit in log
- **WAIT**
  - Wait
  - Any No?
  - Yes: Write abort in log
  - No: Write commit in log
- **COMMIT**
  - Write end of transaction in log

**Participant**
- **INITIAL**
  - Prepare
  - Vote-commit (all)
- **READY**
  - Prepare
  - Vote-abort
  - Global-abort
  - Global-commit
- **ABORT**
  - Write abort in log
  - Write-commit in log
- **COMMIT**
  - Write-commit in log
  - Write-commit in log

**Coordinator**
- **INITIAL**
  - Prepare
  - Vote-commit
- **WAIT**
  - Prepare
  - Vote-abort
- **COMMIT**
  - Ack
  - Ack

**Participants**
- **INITIAL**
  - Prepare
  - Vote-commit
- **READY**
  - Global-abort
  - Global-commit
- **ABORT**
  - Ack
  - Ack
- **COMMIT**
  - Ack
  - Ack

---

**State Transitions in 2PC**

**Coordinator**
- **INITIAL**
  - Commit command
  - Prepare
  - Vote-commit
- **WAIT**
  - Vote-abort
  - Global-abort
- **COMMIT**
  - Ack

**Participants**
- **INITIAL**
  - Prepare
  - Vote-commit
- **READY**
  - Global-abort
  - Global-commit
- **ABORT**
  - Ack
  - Ack
- **COMMIT**
  - Ack
  - Ack
Site Failures - 2PC Termination

- Timeout in INITIAL
  - Who cares
- Timeout in WAIT
  - Cannot unilaterally commit
  - Can unilaterally abort
- Timeout in ABORT or COMMIT
  - Stay blocked and wait for the acks

Site Failures - 2PC Termination

- Timeout in INITIAL
  - Coordinator must have failed in INITIAL state
  - Unilaterally abort
- Timeout in READY
  - Stay blocked
Site Failures - 2PC Recovery

- Failure in INITIAL
  - Start the commit process upon recovery
- Failure in WAIT
  - Restart the commit process upon recovery
- Failure in ABORT or COMMIT
  - Nothing special if all the acks have been received
  - Otherwise the termination protocol is involved

Site Failures - 2PC Recovery

- Failure in INITIAL
  - Unilaterally abort upon recovery
- Failure in READY
  - The coordinator has been informed about the local decision
  - Treat as timeout in READY state and invoke the termination protocol
- Failure in ABORT or COMMIT
  - Nothing special needs to be done
Problem With 2PC

- **Blocking**
  - Ready implies that the participant waits for the coordinator
  - If coordinator fails, site is blocked until recovery
  - Blocking reduces availability

- **Independent recovery is not possible**

However, it is known that:
- Independent recovery protocols exist only for single site failures; no independent recovery protocol exists which is resilient to multiple-site failures.

- So we search for these protocols – 3PC

Network Partitioning

- **Simple partitioning**
  - Only two partitions

- **Multiple partitioning**
  - More than two partitions

- **Formal bounds:**
  - There exists no non-blocking protocol that is resilient to a network partition if messages are lost when partition occurs.
  - There exist non-blocking protocols which are resilient to a single network partition if all undeliverable messages are returned to sender.
  - There exists no non-blocking protocol which is resilient to a multiple partition.
Independent Recovery Protocols for Network Partitioning

- No general solution possible
  - allow one group to terminate while the other is blocked
  - improve availability
- How to determine which group to proceed?
  - The group with a majority
- How does a group know if it has majority?
  - Centralized
    - Whichever partitions contains the central site should terminate the transaction
  - Voting-based (quorum)

Quorum Protocols

- The network partitioning problem is handled by the commit protocol.
- Every site is assigned a vote $V_i$.
- Total number of votes in the system $V$
- Abort quorum $V_a$, commit quorum $V_c$
  - $V_a + V_c > V$ where $0 \leq V_a, V_c \leq V$
  - Before a transaction commits, it must obtain a commit quorum $V_c$
  - Before a transaction aborts, it must obtain an abort quorum $V_a$