Proposal for Replicated Concurrency Control and Recovery Project

Team Members:

- Shika Rao (sr7463)
- Khushboo . (kx2252)

Programming Language: Python and C++

1. Overview

This project implements a distributed database simulation called RepCRec (Replicated Concurrency Control and Recovery). The goal is to model how a distributed system manages data replication, concurrency control, and failure recovery under Serializable Snapshot Isolation (SSI) using the Available Copies algorithm.

The simulated system consists of:

- 10 sites (servers) numbered 1–10.
- 20 variables (x1 …) each initialized to 10 × i.
- Replicated even-indexed variables (x2, x4, …,) stored on all sites.
- Non-replicated odd-indexed variables (x1, x3, …,) stored on exactly one site, determined by
  site = 1 + (index % 10).

Transactions are issued via commands such as begin(T1), R(T1, x3), W(T2, x6, 50), fail(3), and end(T1).
The system processes these inputs and outputs:

- Values read and written
- Transaction commit or abort decisions
- Site activity affected by failures or writes
- Complete database state on dump() commands

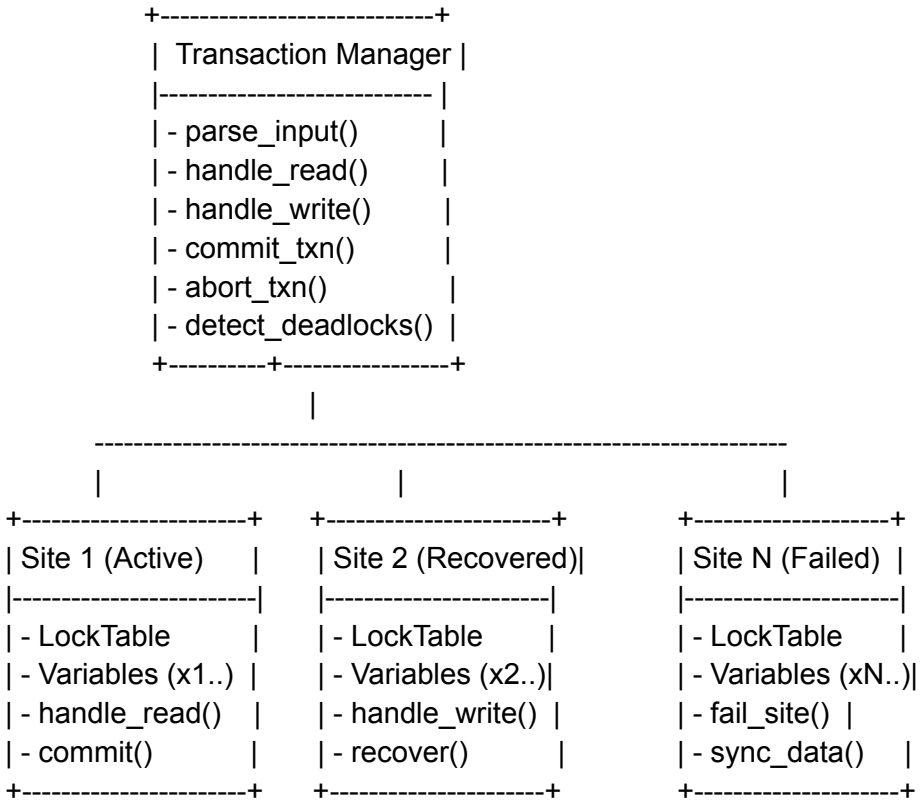
2. System Architecture

- Transaction Manager (TM)
  Coordinates transactions, routing requests, enforcing SSI, handling concurrency and

recovery.

- Data Managers (DMs) (one per site)
  Handle variable storage, version control, local locks, and recovery state.

- Sites
  Represent physical database replicas that can fail or recover.



3. Major Modules

3.1 Transaction Manager (TM)

Purpose:
Coordinates all transactions, manages concurrency and failure information, and enforces serializable snapshot isolation.

Inputs:
Parsed user commands — begin(T), R(T, x), W(T, x, v), end(T), fail(i), recover(i), dump()

Outputs:
Execution events and outcomes:

- Read values (e.g., x3: 120)
- Commit/abort notifications
- Affected sites during writes
- Database states (dump() output)

Responsibilities:

- Maintain a transaction table (status, start time, read/write sets).
- Keep a site status table (up/down state).
- Record dependency edges between transactions for cycle detection.
- Implement first-committer-wins and abort rules from SSI.
- Abort transactions that wrote to a failed site before commit.
- On recovery, restrict reads from replicated variables until new commits occur.

Side Effects: Updates global metadata, transaction states, and initiates site operations.

3.2 Data Manager (DM)

Purpose:
Simulates the data layer at each of the 10 sites. Each DM independently manages variables, their versions, and commit history.

Inputs:
Read and write requests from the TM, along with failure and recovery events.

Outputs:

- Returned data values
- Acknowledgements of successful writes
- Notifications when site is unavailable

Responsibilities:

- Maintain a local datastore mapping variable → (value, commit_time).
- Log site up/down status.
- On failure: clear volatile snapshot information.
- On recovery:
    - Non-replicated variables are immediately available.
    - Replicated variables are unavailable for reads until a new commit.

- Support snapshot reads for transactions (value as of T's start time).
- Apply writes only to available copies during commit.

Side Effects: Local commit metadata and variable values are updated per transaction.

4. Key Data Structures

Data Structure	Description
Transaction Table	Stores each transaction's ID, start time, state (active/committed/aborted), and read/write sets.
Site Table	Tracks whether each site is up or down and timestamps of failures and recoveries.
Version Table	Maintains version history of each variable (value + commit timestamp).
Wait Queue	Contains transactions waiting for unavailable variables or failed sites.
Conflict Graph	Tracks RW and WW dependencies for cycle detection and serialization validation.

5. Concurrency Control and Recovery Rules

Concurrency (SSI):

- Transactions read snapshots valid as of their start time.
- Writes occur on all available copies of the variable.
- At end(T) (commit time):
    - Transaction aborts if any of the following are true:
    - Another transaction committed a write to the same variable since T began (First-Committed-Wins).
    - There are two consecutive RW edges in a dependency cycle.
    - A site written by T failed before T committed.

Failure Handling:

- On fail(i):
  Site i becomes unavailable; its memory of uncommitted writes and SSI state is lost.
  Any transaction that wrote there before commit will later abort.
- On recover(i):
  Site i is marked "up."
  Non-replicated variables can be read/written immediately.
  Replicated variables are writable but not readable until a new commit occurs.

6. Component Interactions

[TM receives command]
↓
[Parse & identify operation]
↓

READ (R)	Query available sites for last committed snapshot value
WRITE (W)	Send write request to all up sites that hold that variable
FAIL/RECOVER	Update site table and adjust affected transactions
END(T)	Validate for conflicts, apply commit or abort

7. Class and Function Design

class TransactionManager

- transactions: dict
- site_status: dict
- waits_for_graph: graph

Funcs:

+ start_transaction(id)
+ handle_read(T, x)
+ handle_write(T, x, value)
+ end_transaction(T)
+ detect_cycles()
+ commit_transaction(T)
+ abort_transaction(T)
+ fail_site(i)
+ recover_site(i)
+ dump_state()

class DataManager

- site_id
- is_active
- variables: map[x] → (value, commit_time)

Funcs:

+ read(x, timestamp)
+ write(x, value, txn_id)
+ fail()
+ recover()

class Variable

- name
- versions: list[(value, commit_time)]
- read_lock, write_lock

class Site

- id
- data_manager
- status