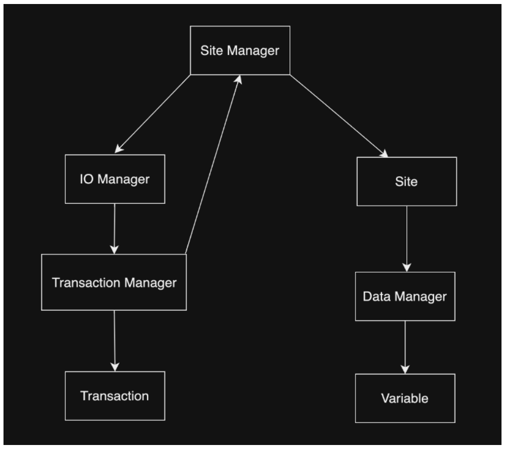
Project Design Doc

Sathvika Bhagyalakshmi Mahesh, Net ID: sb8913
Zihan Zhang, Net ID: zz4412



Site Manager

sites: List of Site objects

failureHistory: To store all failed sites

initialize_sites(self): Initialize values with default, and store it in the corresponding sites

fail(site_id, current_time): Fails the site with the given site_id

recover(site_id, current_time, transaction_manager): Recovers the failed site with the given site_id

IO Manager(file_path)

currentTime: Incrementing integer, to record the time of each instruction

filePointer: Pointer to the input file

get_instruction(): Fetch the next instruction from the file

process_instruction(instruction, transaction_manager, site_manager): Handle each instruction and operate accordingly

dump(site_manager): Print out the committed values of all variables at all sites

close(): Close the filePointer when finish reading

Transaction Manager(site_manager, io_manager)

site_manager: To retrieve info easily from site manager

io_manager: To retrieve info from IO Manager

Transactionclosingcycle: Track of the transaction that closes the cycle

Transactions: Transaction id to Transaction object pointer mappings.

waitlist: HashMap of int, List[int] - key is the site ids which are down and value is the list of transactions blocked on that site waiting for recovery to proceed further

serializationGraph: HashMap of transaction_id, List[depending transactions where edge exists] - Adjacency list of serialization graph, when a dependency is detected the edge is added later checked for cycles

transactionHistory: HashMap of transaction_id, HashMap[variable_id, if R or W was performed] - transaction_id, variable_id, Read/Write is kept track for detecting cycles

get_trans(transaction_id): fetches a particular transaction

begin(transaction_id, timestamp):adding it to Transaction, and adding it in the serializationGraph

read(transaction_id, variable_id): add it to transactionhistory and serializationgraph, checking sites at which variable is available at which site

write(transaction_id, variable_id, value): Store it in history then do the required changes

end(transaction_id): once it ends we abort transactions forming cycles, and commit all changes to the sites

check_for_cycle(start_txn): check if any transaction needs to be aborted due to cycle formation

dfs_cycle_check(node, visited, recStack): helper function for check_for_cycle

Transaction(transaction_id, timestamp)

TransactionStatus(Enum): ONGOING = 1, WAITING = 2, ABORTED = 3, COMMITTED = 4

tid: Transaction ID, passed into class

startTime: Timestamp of start time, passed into class

status: Current status of transaction at a given time from TransactionStatus

isReadOnly: Boolean, flag of read only instructions

Site(site_id)

SiteStatus(Enum): UP = 1, DOWN = 2

status: Site status from SiteStatus. Default is DOWN

id: Site id from 1 to 10, passed into class

data_manager: A site has its own DataManager

get_data_manager(): Initialize a DataManager and return it

Data Manager

committedVariables: to keep track of value of variables at the site committed after end of a transaction

localCopiesPerTransaction: to keep track of all writes but not reflect on the site since transaction is not ended yet, then will commit only after transactions has no conflicts

update_local_copy(transaction_id, variable_id, value): the values are not push to the sites yet, it is like a log storing it temporarily until end of transaction to check for conflicts and the commit changes

commit_transaction(transaction_id, start_time, commit_timestamp): store it in the local_copy until end of transaction

find_most_recent_snapshot(timestamp, variable_id):get the most recent snapshot from the committed variables

get_variables(self): fetch the variable

Variable

variable_name: String, from x1 to x20

value: Value of variable

snapshots: List of Hashmap. Records of timestamp and value pairs.

update_snapshot(current_timestamp, value): Prepend the new snapshot to the snapshots list with the current time

find_snapshot_before_time(timestamp): Return the most recent snapshot before the given timestamp