Advanced DB

Replicated Concurrency Control and Recovery
Design Document

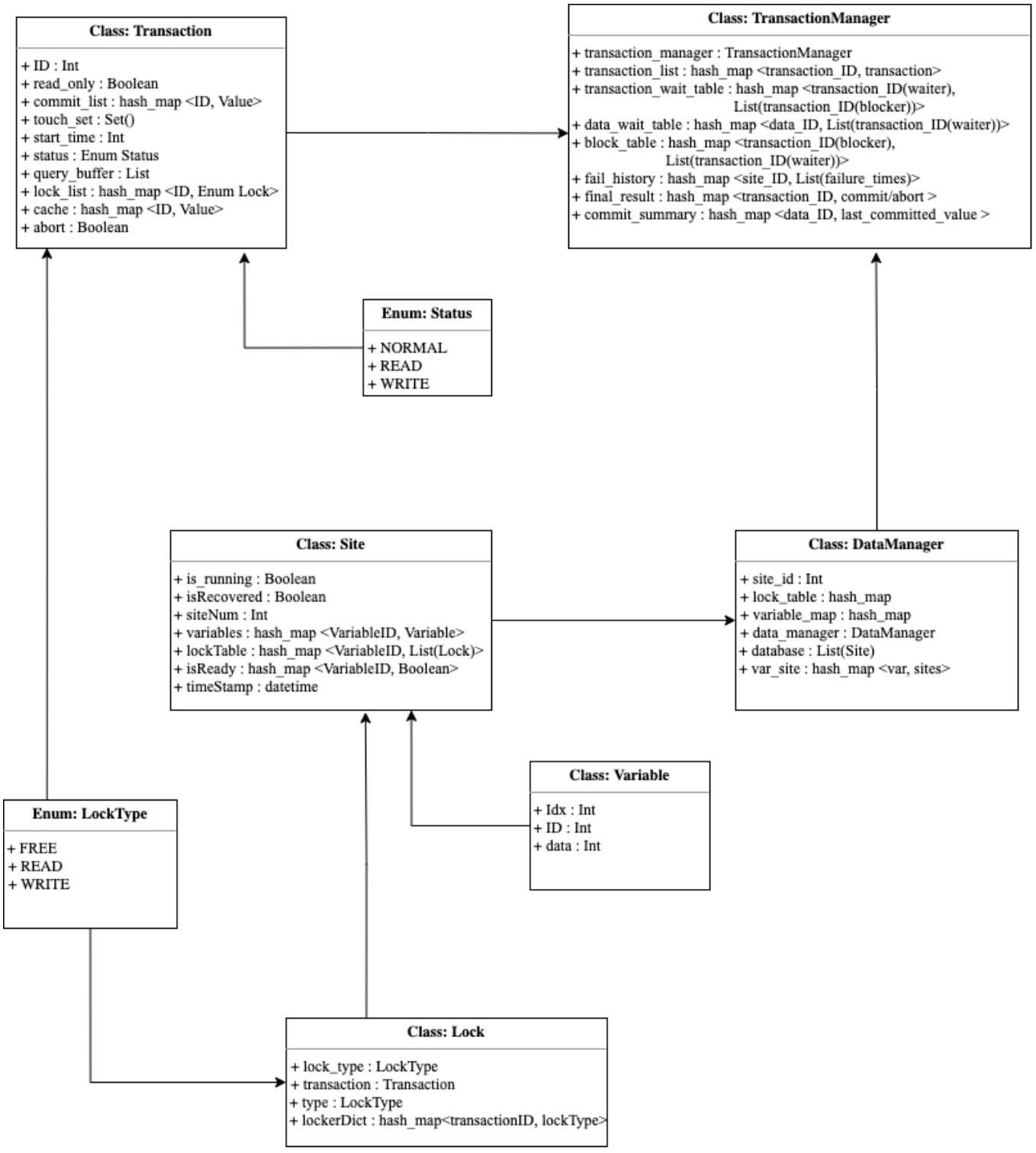

FIG: Class Signatures and dependencies.

PART 1: Transaction Manager

1. transaction.py

   A. **Class Transaction (Safwan Mahmood)**

   def dump() —> Void :
   print transaction info and status

2. transaction_manager.py

   A. **Class TransactionManager (Metarya Ruparel)**

   def parser(input_file) — > None:
   read and parse input files line by line and call corresponding functions

   def begin(transaction_id, time, ro=False) —> None:
   create new transaction

   def read(transaction_id, variable_id, sys_time) —> None:
   attempt to read and call DM.read
   if read success:
   - DM.read will return from which site is touched
   - transaction's lock_list will be updated
   if read fail:
   - DM.read will return which transaction(s) is/are blocking
   - data_wait_table, transaction_wait_table and block_table will be updated

   def write(transaction_id, variable_id, value) —> None:
   attempt to write and call DM.write
   if write success:
   - DM.write will return from which site is touched
   - transaction's lock_list and commit_list will be updated
   if write fail:
   - DM.write will return which transaction(s) is/are blocking
   - data_wait_table, transaction_wait_table and block_table will be updated

   def dump(site=None, variable=None) —> None:
   calls DM.dump

   def fail(site_id, sys_time) —> None:
   - calls DM.fail
   - update fail_history

   def recover(site_id) —> None:
   calls DM.dump

   def end(transaction_id, sys_time) —> None:
   check if transaction should be committed or aborted and call corresponding function

   def commit(transaction_id, sys_time) —> None:
   - called by TransactionManager.end()
   - will call DM.commit and write committed value into available sites
   - then call TransactionManager.release_locks() to release data locked by transaction
   - finally delete committed transaction

def abort(transaction_id, sys_time) —> None:
- called by TransactionManager.end()
- call TransactionManager.release_locks() to release data locked by transaction
- finally delete committed transaction

def deadlock_detection(sys_time) —> None:
- check if there is a cycle in transaction_wait_table
- if cycle: abort the youngest

def release_locks(transaction_id, sys_time) —> None:
- called by commit or abort
- call DM.release_locks and get newly freed data
- distribute newly freed data to transactions in wait

def resurrect(sys_time) —> None:
retry transactions blocked by site failure

def validation(sites_touched, start_time, end_time) —> Boolean:
- check if any site touched by transaction during first read/write to end has failed
- return : True - no site failure during time zone
False - one or more site failure during time zone

def retry(transaction_id, sys_time) —> None:
retry transaction in wait

PART 2: Data Manager

1. data_manager.py

A. **Class DataManager (Safwan Mahmood)**

def generateCacheForRO(trans) —> None:
generate cache for read only transaction

def read(trans, ID) —> (Boolean, Int):
- check if the transaction could get the read lock or not.
- return : (True, siteNum)
: (False, blockers)
: (False, -1), if all the sites which have variables down

def write(transID, ID): —> (Boolean, Int):
- check if the transaction could get the write lock or not.
- return : (True, siteNum)
: (False, blockers)
: (False, -1), if all the sites which have variables down:

def commit(transId, commitList) —-> None:
commit the values for variables in commitList

def writeValToDatabase(ID, val) —> None:
called in commit

def releaseLocks(transID, lockDict) —> Set():
- release locks after commit or abort.

- return : variables which turned Free after the release

def dump(siteNum=None, ID=None) —> None:
- dump the values of variables
- dump all data in the database;
- dump all data in a certain site;
- dump a certain variable in all sites;

def fail(self, siteNum) —> None:
fail a running site

def recover(self, siteNum):
recover a failed site

2. site.py

A. **Class Site (Metarya Ruparel)**

Getters:
def getTime() —> datetime:
return timestamp

def get_all_variables() —> hash_map <VariableID, List(Lock)>:
return variables

def get_variable(ID) —> List(Lock):
return variables[ID]

def get_lock_type(ID) —> Boolean:
return lockTable[ID].type

def get_site_num() —> Int:
return siteNum

def is_up() —> Boolean:
return isRunning

Others:
def is_variable_free(ID) —> Boolean:
return variable with ID free or not

def is_var_valid(ID) —> Boolean:
return whether this variable's value is valid for read

def is_replicated(ID) —-> Boolean:
- used in deciding whether a variable is ready to be read right after recovery
- return whether a variable has replicate in other sites

def lock_var(ID, transID, lockType) —> None:
add a lock to the variable

def unlock(transID, ID) —> None:
remove locks by a certain transaction

def write_var_val(ID, val) —> None:
write val to variable with ID

```
def failSite() —> None:
    fail this site and clear the lock table

def recoverSite() —> None:
    recover the site and initialize the lock table
```

3. lock.py

   A. **Class Lock (Safwan Mahmood)**

```
Getters:
    def is_free() —> Boolean:
        determine if this lock is actually free

    def get_type() —> Enum LockType:
        get the lock type

    def get_locker() —> List(transaction_id):
        get the lockers(transactions)

Others:
    def addLock(transID, lockType) —> None:
        add a lock to this lock object

    def removeLock(transID) —> None:
        remove a lock by transID
```

4. variable.py

   A. **Class Variable (Metarya Ruparel)**

```
def get_ID() —> String:
    return ID
def get_data() —> Int:
    return data
def set_data(newData) —> None:
    set data to newData
```

The flowchart below depicts how a transaction would flow through the system.

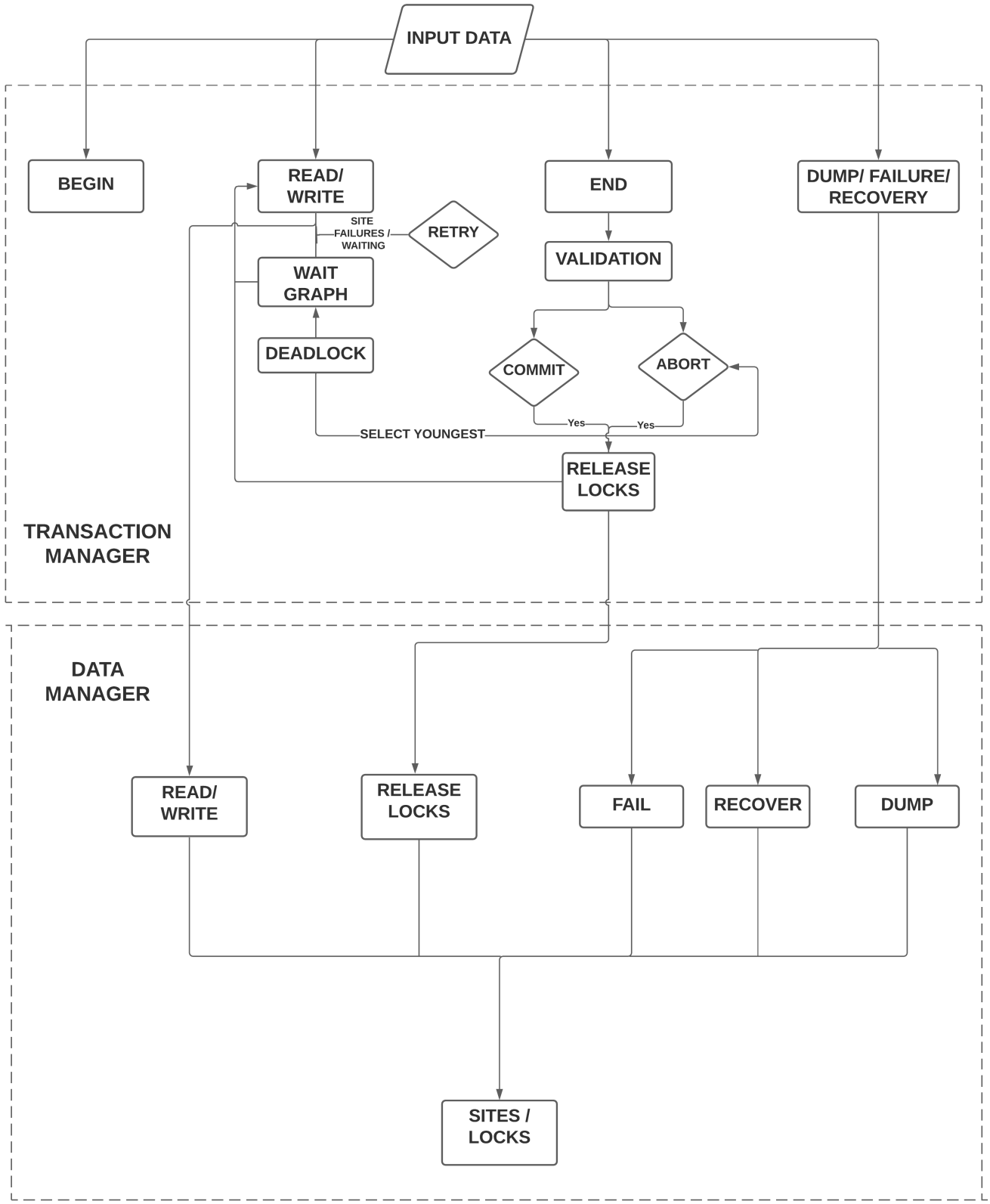

FIG: Flow of transaction through the System.