Design Document for Final Project

Yinqi (Bill) Sun
New York University
ys3540@nyu.edu

Topic: Replicated Concurrency Control and Recovery.

1. INTRODUCTION

In this project, I will implement a mini In-Memory database that handles concurrent transaction processing with recovery. Available copy algorithm and two-phase locking will be used to enhance recoverability. Deadlock detection and prevention algorithms will also be applied in order to make sure the database will not stall on deadlocks.

The database will consist of 2 major components and auxillary functions that helps with illustrating the internals of this database. The details of each module are elaborated in the section below.

2. USAGE

Initiate an interactive session simply by executing

```
python3 io.py
```

Execute single test file by

```
python3 io.py input_file
```

Execute a batch test by

```
python3 io.py mega directory_to_test_files
```

Additional commands for **Interactive session**:

- source input_file: to execute an input file during interactive session
- mega directory_to_tests: to execute an input file during interactive session
- reset: reset the database (TM and DM)
- hi: will reply 'hello'. Can be used to test whether the program freezes
- exit: quit the session

3. MAJOR MODULES

The architecture can be illustrated on figure 3.3.

3.1 Transaction Manager

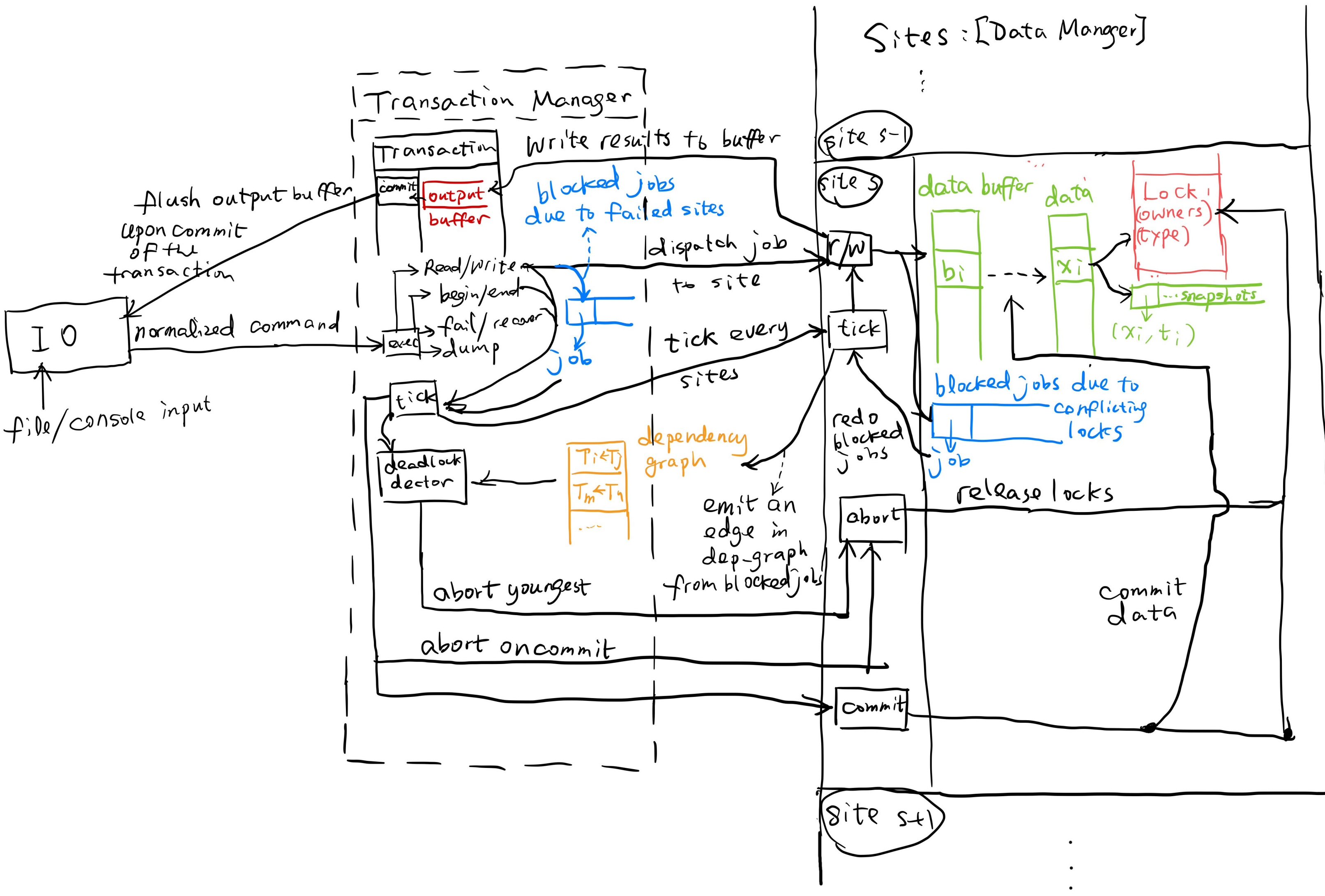
A global Transaction Manager (TM) keeps track of the availabilities of each site. It accepts transactions from the input, tag them with a timestamp and issue read write **jobs** to corresponding sites according to the available copy algorithm. It decides whether a job should block, success or fail according to the availability of sites and the execution result from (each) site. During transactions, it's also responsible for detecting deadlocks periodically and resolving it by dropping the transaction with largest timestamp. At every tick, the TM will repeatedly tick each data manager to retry blocked jobs and commit or abort transactions whenever there're no more jobs left for that transaction (an end(Ti) is issued) until there're no change to the blocked queue. Then, it will detect circles in the dependency graph and abort youngest transaction immediately.

3.2 Data Manager

I will also emulate distributed storage by keeping data items in ‘sites’. Each site has it’s independent storage and a Data Manager (DM) that is responsible for taking read and write jobs from Transaction Manager and act accordingly. The data manager will maintain a lock table that keeps a **Lock** object for each data item. Each lock object will record the lock status and the owner(s) of the lock. There’re four different lock state, 0 means unlocked, 1 means a read (shared) lock is acquired, 2 indicate a write lock has been acquired and 4 means that the object was only available for write operations due to the site’s recent recovery.
Each DM also maintain a blocked queue that registers the blocked operations and may retry them at every tick. At the end of a tick, the data manager will update the global dependency graph by emitting an edge (Ti, Tj) whenever there are a lock dependency from Tj to Ti.

3.3 Auxillary functionalities

There’re other auxillary functions for interaction, debugging or illustration purposes. such as a Command Parser that parses the input command and feed them to Transaction Manager. An output manager that’s responsible for generate formatted outputs and error messages.



Basic Design Graph