RepCRec Design Document

Siddhant Meshram - sm10954@nyu.edu Sujana Maithili Chindam - sc10648@nyu.edu

Code Structure:

We will be implementing the Replicated Concurrency Control and Recovery (RepCRec) project. We use C++ for the implementation. Please find below the high-level details of the classes we created and the functions within these classes.

- **Transaction Manager Class:** (authors: Siddhant, Sujana)

  The Transaction Manager is the top level class that manages all the transactions and maintains Site data which is used while processing the transactions. The transactions data is maintained as a struct within this class. This includes fields such as begin_time, end_time, commit_time, and list of all sites transaction has accessed for reads and wrties, etc. All the sites are also maintained in a data structure within this class. Additionally, we also maintain the serialization graph within this class.

  - **processInput:** Processes the input one line at a time and provides output if required by the command.
  - **processBegin(transaction_name):** Initialises a new transaction instance and adds the transaction to the active transaction list.
  - **processRead(variable):** Handles all the read requests and does all the checks required before processing the read. Aborts if the site is not safe to read. Also, waits for a site to recover if no site is available for reading.
  - **processWrite (variable):** Handles all the write requests. Before commit, all writes are done locally to each site. Also, waits for a site to recover if no site is available for writing.
  - **isSafeToCommit (transaction_name):** At the end of a transaction, runs all the checks for that transaction and determines if it is safe or unsafe to commit.
    - **check_available_copies_safe:** Checks if no site that the transaction accessed went down after the transaction began.
    - **check_snapshot_isolation_safe:** Checks if any other transaction modified the same data variable on the same site after the transaction began.
    - **Check_seralizability_safe:**
      - **add_transaction_to_graph:** Adds the transaction to the serialization graph by adding ‘ww’, ‘wr’, and ‘rw’ edges. These edge types are decided based on the access types of the variables and the commit/begin times of other relevant transactions.
      - **hasTwoRwCycle():** Checks if there is a cycle with two consecutive rw edges. To do so, we implemented a backtracking algorithm that detects all the cycles in graphs (implementation in function **detectCycle**) and

then analyze to check if the detected cycle has consecutive rw edges (implementation in function **analyzeCycle**)

.

- **processCommit (transaction_name):** A transaction is committed if isSafeToCommit returns true. This also commits the local writes of the transaction.
- **processAbort (transaction_name):** A transaction is aborted if it is not commit safe. The aborted transaction edges from the serialization graph are removed.
- **processRecover (site):** It can happen that there are some operations waiting for a site to recover. In such cases, when the site recovers, the waiting requests will be processed. The waiting requests can be both read or write.

- **Site Class:** (authors: Siddhant, Sujana)

  Each site maintains all the states relevant to a site. It stores all the variables for that site and also has a Data Manager that manages all the variables in that site.

  - **readData (variable, transaction_name):** This function is called by the processRead function of the Transaction Manager class given it meets the safety checks in processRead. It calls the getValue function of the Data Manager Class.
  - **writeLocal (variable, transaction_name, value):** This function is called by processWrite of Transaction Manager class. This function will only write the variable locally and not commit the data. It calls the writeLocal function of the Data Manager Class.
  - **commitData (transaction_name):** This function is called by the processCommit function of the Transaction Manager class. For the given transaction, we get all the variables to commit, and call the commitData function of the Data Manager Class on each variable.

  Other functions such as isUp(), setUp(), last_down(), setDown() are also implemented.

- **Data Manager Class:** (authors: Siddhant, Sujana)

  Data Manager is used to manage the data stored on a particular site.

  - **getLastCommittedTimestamp (variable):** Returns the committed timestamp of the variable at that site.
  - **getLastCommittedTimestamp (variable, time):** Returns the committed timestamp of the variable at that site before “time”.
  - **getValue(variable, transaction_name):** If this transaction has made a local write then it returns the locally written value. Otherwise, returns the value that was committed.
  - **writeLocal (variable, transaction_name, value):** This function will only write the variable locally and will not commit the data for the given transaction.
  - **commitData (variable, value):** Commits the locally written data for the variable.

Steps to run the program:

````
To compile the code, you can run `make`.

After compiling, you can run the code by using the following command and
providing input using stdin.
```
./repcrep
```

If you want to provide the input from a file, you can use input
redirection.
```
./repcrep < <input_file>
```

If you want to run all the tests under inputs directory, you can use the
script provided:
```
./script.sh
```
````

With Reprozip:

````
We have provided a repro-test.rpz file which is created using reprozip. It
can be unzipped using

```
reprounzip directory setup repro-test.rpz ./unzipped_dir
```

To run the executable:
```
reprounzip directory run ./unzipped_dir
```
To pass input as a file:
```
reprounzip directory run ./unzipped_dir < <input_file>
```
````

Testing performed:

The inputs directory contains all the inputs on which we tested our code. We tested our code on the test cases provided to us and some of our own test cases. We generated test cases input26-input40 which can be found in the inputs directory. The description of these test cases can be found in these input files.

Design Diagram: