School of Computer Science, McGill University

COMP-512 Distributed Systems, Fall 2018

Project Milestone 3: 2-Phase Commit

Report Date: 6 PM before your demo, Demo date: November 29/30

In this last phase of the project you will implement the 2-phase commit protocol in your system. You may assume that communication is reliable, but as noted in the following sections, servers may crash at various points of the protocol.

Question 1: *Implementation* (90 Points)

1. **Implement data persistence using shadowing**. Each `ResourceManager` (and if applicable, the `Middleware`) should have two files: one committed version and one for transactions in-progress. You should also have an associated master record for each `ResourceManager`. On startup, the above files should be initialized if they are not already present.

2. **Implement basic 2-phase commit**. For the first iteration of your design, implement `commit` and `abort` assuming failures do not happen. In this part, you should design your logging architecture, and when/what to write to disk. In particular, ensure that a transaction in the prepared state may still be either committed or aborted.

   In our system architecture, the `TransactionManager` is the coordinator.

3. **Implement the crash API**. Described in the following section is a standard API for triggering specific types of failures in your system. You should modify your client with the additional commands and add `System.exit(1)` calls at the corresponding places in your server implementation if the crash mode is enabled.

4. **Implement failure handling**. Failure handling can be broken in two main sections:

   (a) **Implement failure detection**. All failures in the system (client, `ResourceManagers`, `Middleware`, etc.) should be detected and handled accordingly. You may need to detect such errors via a timeout mechanism in your protocol. Note that in RMI, an exception is thrown if the remote method crashes - depending on your implementation it may be beneficial to catch and handle these exceptions.

   Remember, the only case where a transaction should block indefinitely (and not timeout) is when a participant votes YES and does not receive any message from the transaction coordinator. In all other cases, the transaction should eventually commit/abort. You do not need to implement a termination protocol to "ask around" when uncertain. Be sure to handle cases where the timeout is premature - e.g. a participant times out prematurely and locally aborts its transaction before the coordinator.

   (b) **Implement recovery**. For this project you may implement the recovery procedure for either the `Middleware` or `ResourceManager`.

5. **Display appropriate output**. You should display enough useful output on `stdout` to convince us that your system is working as intended.

Interface Requirements

At the `ResourceManagers`, add the initiation method for the 2-phase commit protocol:

```
public boolean prepare(int xid)
throws RemoteException, TransactionAbortedException, InvalidTransactionException;
```

At the `TransactionManager` and `ResourceManagers`, add 3 `crash` methods to trigger/disable failures in your system.

- **Reset crashes** (disable all):
  ```
  public void resetCrashes() throws RemoteException;
  ```
- **TransactionManager crashes**:
  ```
  public void crashMiddleware(int mode) throws RemoteException;
  ```
- **ResourceManager crashes**:
  ```
  public void crashResourceManager(String name /* RM Name */, int mode)
  throws RemoteException;
  ```

Crash Modes

You should implement the following crash modes, corresponding to the `mode` parameter of the above methods. You will be asked to demonstrate a subset to the TA.

At the TransactionManager (coordinator):

1. Crash before sending vote request
2. Crash after sending vote request and before receiving any replies
3. Crash after receiving some replies but not all
4. Crash after receiving all replies but before deciding
5. Crash after deciding but before sending decision
6. Crash after sending some but not all decisions
7. Crash after having sent all decisions
8. Recovery of the coordinator (if you have decided to implement coordinator recovery)

At the ResourceManagers (participants):

log YES before

1. Crash after receive vote request but before sending answer
2. Crash after decigin which answer to send (commit/abort) after YES, before return before abort
3. Crash after sending answer after return
4. Crash after receiving decision but before committing/aborting begin of commit abort
5. Crash during recovery begin of restart

Custom Functionality

This milestone again requires some type of creative functionality depending on your interests. You may also email the TA with any other ideas you may have for approval – use your creativity!

- Subtransactions recovery - if this was implemented in the previous milestone
- Full recovery of both the `ResourceManagers` and `Middleware`
- Protocol for "asking around" on failure recovery
- Creative functionality - ask the TA for confirmation before!

Demo

The final project demo will cover all aspects of your system from the beginning of the semester up to and including this part. You should therefore spend some time fixing issues from previous demos if possible, using the feedback from myCourses. Since this section of the project has more to cover, the demos will last 30 minutes, and the milestone will contribute more to your overall grade.

The final demo will start with a brief discussion of your implementation decisions, and continue with a live testing of your project. You must have your system up and running when it begins, and be prepared to answer questions about your design. We will test communication/distribution, transactions, locking, and failures. The source code must be submitted to myCourses by the end of the demo period.

Question 2: *Report* (10 Points)

The last report of this project is slightly longer, and will cover all aspects of your system. You may re-use previous reports - however it must be cohesive and updated following any design changes.

- The general design and architecture of your system (high-level details)
- How each of the individual features of the system is implemented
  - Distribution and communication
  - Transactions and locking
  - Data shadowing
  - Logging and recovery (2-PC)
- Special features of your implementation
- Problems that you encountered
- How you tested the system for correctness

You report should be around 5-6 pages - try to be concise, but cover all main points.