Distributed Systems

IN5020

Programming Assignment 2

shielak, praveema, kathabar

Contents

1 User guide to compiling and running the distributed application

steps:
1) The first step is to compile the java files.

```
javac *.java
```

2) run the AccountReplica for n - 1 times where n describes the number of replicas you want to use

```
java AccountReplica 127.0.0.1 myaccount n
```

3) run AccountReplica one more time like before if you want to enter the commands manually// or with the filename if you want to use a input file.

```
option 1 ) java AccountReplics 127.0.0.1 myaccount n
option 2) java AccountReplics 127.0.0.1 myaccount n input.txt
```

4) if you want to test the naive implementation, change line 38 in AccountReplica to ‘true’

```
private static boolean naive = true;
```

2 Design Choices

2.1 High Level Algorithm

1. Balance is set as 0 and all other variables are initialized
2. Each replica waits for ‘n’ number of replicas to join
3. When replica receives filename as one of the command line arguments it acts as primary
4. when replica sees the group contains ‘n’ replicas it sync the balance with other replicas if it joins the group later
5. Primary:
    a. parses the command from file one by one
    b. if the command is deposit or addinterest, then it adds the command to the outstanding collection
    c. if the command is getSyncedBalance, then it handles naive or advanced version based on the configuration
    d. it handles all other commands based on the logic described in the assignment
    e. it runs the executor service and calls the run method every 10 seconds which multicasts the outstanding collection

2.2 Listeners

1. when replica receives new spread message from spread connection, the listener for is invoked
2. If the command is deposit or addInterest, then it invokes the corresponding methods and updates the balance
3. The transaction is removed from the outstanding collection and increments the counter
4. The transaction is added to the executed list and increments the counter

2.3 GetSynced Balance (naive and advanced version)

naive version:

1. it checks whether the outstanding collection is empty or not
2. if it is not empty then it waits till it becomes empty (when listener removes transaction from the outstanding collection it notifies all other threads waiting for it)
3. if is empty then it prints the balance

advanced version:

1. Primary:
    a. adds the getSyncedBalance command to the outstanding collection
2. Listener:
    a. parses the getSyncedBalance command
    b. since the FIFO multicast communication protocol is used when listener receives this command, it means that it should have received all other previous deposit and addInterst commands
    c. so the listener just prints the balance

Differences in both implementation:

In the naive implementation there is a possibility, that when reading commands from a file the getSyncedBalance result can inherit a transaction which is made directly after the getSyncedBalance balance, as the thread does not lock after after calling this command and the outstanding collection is send after max. 10 seconds. In our tests the chance is very small, but it is theoretically possible. In the advanced implementation we avoid this phenomenon by adding the command to the outstanding collection and as we use FIFO we can make sure that it will be in the correct order.

2.4 Object synchronization between threads

1. Each Replica runs different threads (Main execution thread to handle the normal flow, executor service uses one thread to multicast the outstanding collection periodically, listener for spread messages uses separate thread)
2. Some of the data structures are manipulated by multiple threads concurrently
3. so these data structures are protected by using wait and notify and synchronized mechanisms

2.5 State Sync between replicas

1. when a replica joins it checks whether the group contains enough replicas
2. if group contains enough replicas then it makes sure that it has the updated balance
3. so it sends getState command to other replicas by multicasting it
4. Listener:
    a. when it receives getState command, it multicasts it's balance as stateInfo command
    b. when a replica receives stateInfo command, it updates its balance if the incoming balance is more than its current balance
5. there is a room for bug
    a. when a newly joint replica receives the outstanding collection before the state info from the primary then it can process the outstanding collection first then set the balance which is not consistent. we hope that it won't happen all the time if the group contains more replicas which can send the state info because only primary will send the outstanding collection.

3 Group work

For this assignment we get used to Spread alone first. After that we discussed our difficulties and helped each other. On our second meeting we discussed the assignment text and decided, that this kind of assignment makes it more difficult to split parts of it and work on these parts alone as in the last assignment. This is why we solved this assignment together in a pair programming alike solution. We met in 6 sessions and on a rotating basis each one of us shared the screen and wrote the code, while discussing the possible solutions in the group. We decided for this method because our main logic happens in one file, which makes it even with git more difficult to avoid merge conflicts. Another reason is that Spread Toolkit is just barely documented, so it was the easiest to solve issues directly within the group.