项目技术报告

20210240009 许元武

成员及分工

20210240009 许元武 ： 负责 TransactionManager 和部分测试用例的编写
20210240007 李炳嘉 ： 负责 WorkflowController、实体类和部分测试用例的编写

两阶段提交协议的实现

主要的实现部分都在 TransactionManagerImpl 类中的 commit 方法中。
当 WorkflowController 调用 TransactionManager 的 commit 方法时，TransactionManager 首先会向参与该事务的所有 ResourceManager 发送 prepare 消息。当收到来自所有 ResourceManager 的确认消息后，TransactionManager 再向所有的 ResourceManager 发送 commit 命令，并记录该事务为 finalize 状态，并从当前活跃的事务队列中剔除。
若 TransactionManager 在调用某个 ResourceManager 的 prepare 方法时发生问题（某个 ResourceManager down 了），TransactionManager 会向参与事务的其他 ResourceManager 发送 abort 命令，将该事务回滚。
主要的代码如下：

```
public boolean commit(int xid) throws RemoteException,
InvalidTransactionException, TransactionAbortedException {
   if (!xid2state.containsKey(xid)) {
      throw new TransactionAbortedException(xid, "bad xid");
   }

   // write the prepare log to disk and mark xid PREPARED
   xid2state.put(xid, TransactionState.PREPARED);
   persist(TM_XID2STATE_LOG, xid2state);

   Set<ResourceManager> resourceManagers = xid2rms.get(xid);
   for (ResourceManager resourceManager : resourceManagers) {
      try {
         boolean prepared = resourceManager.prepare(xid);
         if (!prepared) {
            abort(xid);
            throw new TransactionAbortedException(xid, "one rm not
prepared");
         }
```

```
      } catch (Exception e) {
        // occur when RM die before prepare
        abort(xid);
        throw new TransactionAbortedException(xid, "one rm down
before prepared detected");
      }
    }
    if (dieTime.equals("BeforeCommit")) {
      dieNow();
    }

    // write the prepare log to disk and mark xid COMMITTED
    synchronized (xid2state) {
      xid2state.put(xid, TransactionState.COMMITTED);
      persist(TM_XID2STATE_LOG, xid2state);
    }
    if (dieTime.equals("AfterCommit")) {
      System.out.println("going to die...");
      dieNow();
    }

    Set<ResourceManager> committedRMs = new HashSet<>();
    for (ResourceManager resourceManager : resourceManagers) {
      try {
        System.out.println("send commit to rm: " +
resourceManager.getID());
        resourceManager.commit(xid);
        committedRMs.add(resourceManager);
      } catch (Exception e) {
        System.out.println("one rm down before commit detected");
      }
    }
    if (committedRMs.size() == resourceManagers.size()) {
      xid2rms.remove(xid);
      persist(TM_XID2RMS_LOG, xid2rms);

      // all rm committed, transaction finalize
      xid2state.remove(xid);
      persist(TM_XID2STATE_LOG, xid2state);
    } else {
      synchronized (xid2rms) {
        resourceManagers.removeAll(committedRMs);
        xid2rms.put(xid, resourceManagers);
        persist(TM_XID2RMS_LOG, xid2rms);
```

```
        }
    }
    return true;
}
```

异常的处理

ResourceManager 宕机

- 若某 resourceManager 在 client commit 之前就宕机了，那么 TransactionManager 就收不到该 ResourceManager 对 prepare 的反馈，则 client 的这次 commit 会失败，该事务会被回滚。
- 若某 ResourceManager 在收到 TransactionManager 发来的 prepare 消息后宕机，那么 TransactionManager 就收不到该 ResourceManager 对 prepare 的反馈，则 client 的这次 commit 会失败，该事务会被回滚。
    - 若某 ResourceManager 在执行 TransactionManager 发来的 abort 命令时宕机，此时该事务已经被回滚，该 ResourceManager 会在重启之后检查本地记录的事务日志，并向 TransactionManager 发送消息查询其所参与的事务的状态。在得知该事务是 aborted 状态后，该 ResourceManager 会自行回滚事务在本地的操作。
    - 若某 ResourceManager 在执行 TransactionManager 发来的 commit 命令时宕机，此时该事务已经被记录为 committed 状态，该 ResourceManager 会在重启之后检查本地记录的事务日志，并向 TransactionManager 发送消息查询其所参与的事务的状态。在得知该事务是 committed 状态后，该 ResourceManager 会自行将该事务的操作在本地持久化。

TransactionManager 宕机

- 若 TransactionManager 在 prepare 之后、commit 之前宕机了，那么当它重新启动、查看从磁盘中恢复的事务日志时发现该事务还没有被 commmit，会将该事务记为 aborted 状态。同时，参与事务的 ResourceManager 会周期性地尝试重新连接 TransactionManager，在成功重新连接后，得知 TransactionManager 将该事务记为了 aborted 状态，会自行回滚事务在本地的操作。
- 若 TransactionManager 在 commit 后、但还没有收到所有 ResourceManager 成功 commit 的消息前宕机了，那么当它重新启动、查看从磁盘中恢复的事务日志时发现该事务处在 commmit 状态，会等待来自 ResourceManager 的重新连接。参与事务的 ResourceManager 在成功重新连接后，得知 TransactionManager 将该事务记为了 committed 状态，会自行将该事务的操作在本地持久化，完成事务的 commit。

死锁异常

当事务二需要申请事务一占有的锁时（事务二申请对象的写锁或事务二申请对象的读锁同时事务一占有对象的写锁），ResourceManager 会抛出死锁异常，同时会通知调用者要 abort 事务二。

处理宕机异常的代码主要在 TransactionManagerImp 中的 enlist 方法中，如下：

```
public void enlist(int xid, ResourceManager rm) throws
RemoteException, InvalidTransactionException {

   if (!xid2rms.containsKey(xid)) {
      rm.abort(xid);
      return;
   }
   synchronized (xid2state){
      TransactionState xState = xid2state.get(xid);

      // rm down before commit then recover
      if (xState.equals(TransactionState.ABORTED)){
         rm.abort(xid);
         return;
      }
      else if (xState.equals(TransactionState.COMMITTED)) {
         // rm down before commit then recover
         rm.commit(xid);
         System.out.println("receive enlist from: " + rm.getID());

         synchronized (xid2rms) {
            Set<ResourceManager> rms = xid2rms.get(xid);
            rms.remove(rm);
            System.out.println(xid + "now has "+ rms.size() + "rms" );
            // tm receive ack from all rms
            if (rms.size() == 0) {
               xid2rms.remove(xid);
               persist(TM_XID2RMS_LOG, xid2rms);

               xid2state.remove(xid);
               persist(TM_XID2STATE_LOG, xid2state);
            } else {
               persist(TM_XID2RMS_LOG, xid2rms);
            }
         }
         return;
      }else{
```

```
    synchronized (xid2rms) {
      Set<ResourceManager> rms = xid2rms.get(xid);
      rms.add(rm);
      persist(TM_XID2RMS_LOG, xid2rms);
    }
  }
```