Lab 5: Distributed Transaction

1. Cấu trúc

Project được tổ chức theo mô hình client-server với giao thức **2-Phase Commit (2PC)**. Gồm các thành phần chính:

- **client.go**: Client gửi yêu cầu giao dịch đến Coordinator.
- **coordinator.go**: Quản lý quy trình 2PC, điều phối các Server.
- **server.go**: Mỗi server quản lý tài khoản ngân hàng, thực hiện Prepare/Commit/Rollback.
- **transaction.go**: Định nghĩa các struct cho request và response của giao dịch.

2. Nội dung

Thực hiện chuyển tiền từ **account_1** sang **account_2** trên 2 máy chủ khác nhau

1. Client (`client.go`)

- Kết nối đến **Coordinator** qua RPC.
- Gửi yêu cầu giao dịch với thông tin: tài khoản nguồn, tài khoản đích, số tiền.
- Nhận phản hồi từ Coordinator và in kết quả giao dịch.

```
go run client/client.go localhost:50053 account_1 account_2 30
Transaction Result: Transaction committed successfully
```

2. Coordinator (`coordinator.go`)

- **Phase 1: Prepare**
  - Gửi yêu cầu `PrepareTransaction` đến cả hai Server.
  - Nếu một trong hai thất bại, gửi yêu cầu `RollbackTransaction` đến tất cả Server liên quan.
- **Phase 2: Commit**

    - Nếu tất cả Server đã **Prepare thành công**, gửi yêu cầu `CommitTransaction`.
    - Nếu Commit thất bại ở một server, rollback giao dịch.
- **Rollback**
    - Nếu `PrepareTransaction` hoặc `CommitTransaction` thất bại, thực hiện rollback ngay lập tức.

```
func (c *Coordinator) TransferMoney(req models.TransactionRequest, res *mode
    log.Printf("[Coordinator] Starting 2PC transaction: %s → %s, Amount: %d", rec

    fromServer := c.server1
    toServer := c.server2

    if req.From == "account_2" {
        fromServer = c.server2
        toServer = c.server1
    }

    // Phase 1: Prepare
    log.Printf("[Coordinator] Requesting Prepare from %s", req.From)
    err := fromServer.Call("Server.PrepareTransaction", req, res)
    if err != nil || !res.Success {
        log.Printf("[Coordinator] ERROR: Prepare failed for %s, rolling back...", req.F
        fromServer.Call("Server.RollbackTransaction", req, res)
        return err
    }

    log.Printf("[Coordinator] Requesting Prepare from %s", req.To)
    err = toServer.Call("Server.PrepareTransaction", req, res)
    if err != nil || !res.Success {
        log.Printf("[Coordinator] ERROR: Prepare failed for %s, rolling back...", req.T
        fromServer.Call("Server.RollbackTransaction", req, res)
        toServer.Call("Server.RollbackTransaction", req, res)
```

```
        return err
    }

    // Phase 2: Commit
    log.Printf("[Coordinator] Requesting Commit from %s", req.From)
    err = fromServer.Call("Server.CommitTransaction", req, res)
    if err != nil || !res.Success {
        log.Printf("[Coordinator] ERROR: Commit failed for %s, rolling back...", req.F
        toServer.Call("Server.RollbackTransaction", req, res)
        return err
    }

    log.Printf("[Coordinator] Requesting Commit from %s", req.To)
    err = toServer.Call("Server.CommitTransaction", req, res)
    if err != nil || !res.Success {
        log.Printf("[Coordinator] ERROR: Commit failed for %s, rolling back...", req.T
        fromServer.Call("Server.RollbackTransaction", req, res)
        return err
    }

    log.Printf("[Coordinator] SUCCESS: Transaction completed!")
    return nil
}
```

3. Server (`server.go`)

- **PrepareTransaction**
  - Kiểm tra số dư tài khoản nguồn.
  - Nếu hợp lệ, lưu trạng thái "prepared" vào biến **prepared map**.

```
func (s *Server) PrepareTransaction(req models.TransactionRequest, res *mo
    log.Printf("[Server %s] Preparing transaction: %s → %s, Amount: %d", s.a

    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second
    defer cancel()
```

```
    coll := s.client.Database(database).Collection(collection)

    // Kiểm tra số dư tài khoản nguồn
    if req.From == s.accountID {
        var sender struct {
            Balance int `bson:"balance"`
        }
        err := coll.FindOne(ctx, bson.M{"_id": req.From}).Decode(&sender)
        if err != nil {
            res.Success = false
            res.Message = "Source account not found"
            log.Printf("[Server %s] ERROR: Source account not found!", s.account
            return err
        }
        log.Printf("[Server %s] Current balance of %s: %d", s.accountID, req.Fro

        if sender.Balance < req.Amount {
            res.Success = false
            res.Message = "Insufficient balance"
            log.Printf("[Server %s] ERROR: Insufficient balance in %s", s.accountI
            return fmt.Errorf("insufficient balance")
        }
    }

    // Đánh dấu giao dịch đã được chuẩn bị nhưng chưa commit
    s.prepared[req.From] = req.Amount
    log.Printf("[Server %s] Transaction prepared: %s → %s, Amount: %d", s.a

    res.Success = true
    res.Message = "Transaction prepared successfully"
    return nil
}
```

- **CommitTransaction**

- Nếu tài khoản nguồn đủ tiền, thực hiện trừ tiền và cộng tiền vào tài khoản đích.
- Nếu cộng tiền vào tài khoản đích thất bại, rollback ngay lập tức.

```
func (s *Server) CommitTransaction(req models.TransactionRequest, res *mo
    log.Printf("[Server %s] Committing transaction: %s → %s, Amount: %d", s.

    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second
    defer cancel()

    coll := s.client.Database(database).Collection(collection)

    // Thực hiện giao dịch
    _, err := coll.UpdateOne(ctx, bson.M{"_id": req.From}, bson.M{"$inc": bson
    if err != nil {
        res.Success = false
        res.Message = "Failed to debit source account"
        log.Printf("[Server %s] ERROR: Failed to debit %s", s.accountID, req.Fro
        return err
    }

    _, err = coll.UpdateOne(ctx, bson.M{"_id": req.To}, bson.M{"$inc": bson.M
    if err != nil {
        // Nếu cộng tiền vào tài khoản đích thất bại, rollback ngay
        log.Printf("[Server %s] ERROR: Failed to credit %s, rolling back...", s.acc
        s.RollbackTransaction(req, res)
        return err
    }

    // Xóa trạng thái chuẩn bị
    delete(s.prepared, req.From)

    res.Success = true
    res.Message = "Transaction committed successfully"
    log.Printf("[Server %s] SUCCESS: Transaction committed!", s.accountID)
```

```
    return nil
}
```

- **RollbackTransaction**
  - Hoàn trả số dư tài khoản nguồn nếu giao dịch chưa commit.
  - Xóa trạng thái "prepared" khỏi danh sách giao dịch chờ xử lý.

```
func (s *Server) RollbackTransaction(req models.TransactionRequest, res *m
    log.Printf("[Server %s] Rolling back transaction: %s → %s, Amount: %d", s

    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second
    defer cancel()

    coll := s.client.Database(database).Collection(collection)

    // Kiểm tra xem giao dịch có trong danh sách `prepared` không
    if amount, exists := s.prepared[req.From]; exists {
        // Hoàn trả số tiền đã trừ trước đó
        _, err := coll.UpdateOne(ctx, bson.M{"_id": req.From}, bson.M{"$inc": bs
        if err != nil {
            res.Success = false
            res.Message = "Rollback failed"
            log.Printf("[Server %s] ERROR: Rollback failed for %s", s.accountID, re
            return err
        }

        // Xóa khỏi danh sách `prepared`
        delete(s.prepared, req.From)
        log.Printf("[Server %s] Rollback successful!", s.accountID)

        res.Success = true
        res.Message = "Transaction rolled back successfully"
    } else {
        log.Printf("[Server %s] WARNING: No prepared transaction found to roll
        res.Success = false
```

```
        res.Message = "No prepared transaction found"
    }
    return nil
}
```

```
go run server/server.go mongodb://localhost:27017 50051 account_1
2025/03/20 20:42:02 [Server account_1] Running on port 50051, connected t
2025/03/20 20:42:41 [Server account_1] Preparing transaction: account_1 → a
2025/03/20 20:42:41 [Server account_1] Current balance of account_1: 100
2025/03/20 20:42:41 [Server account_1] Transaction prepared: account_1 → a
2025/03/20 20:42:41 [Server account_1] Committing transaction: account_1 →
2025/03/20 20:42:41 [Server account_1] SUCCESS: Transaction committed!
...
```

```
go run server/server.go mongodb://localhost:27018 50052 account_2
2025/03/20 20:42:20 [Server account_2] Running on port 50052, connected
2025/03/20 20:42:41 [Server account_2] Preparing transaction: account_1 →
2025/03/20 20:42:41 [Server account_2] Transaction prepared: account_1 →
2025/03/20 20:42:41 [Server account_2] Committing transaction: account_1 →
2025/03/20 20:42:41 [Server account_2] SUCCESS: Transaction committed!
...
```

4. Database

```
mongosh --port 27017
Current Mongosh Log ID: 67dc1a3565af88b8fab71235
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverS
Using MongoDB:          8.0.5
Using Mongosh:          2.4.2
```

```
For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-03-20T20:37:48.623+07:00: Access control is not enabled for the databa
------

test> use bank
switched to db bank
bank> db.accounts.insertOne({_id: "account_1", balance: 100})
{ acknowledged: true, insertedId: 'account_1' }
bank> db.accounts.find()
[ { _id: 'account_1', balance: 100 } ]
bank> db.accounts.find()
[ { _id: 'account_1', balance: 70 } ]
bank>
```

```
mongosh --port 27018
Current Mongosh Log ID: 67dc1a4c7cb2769f86b71235
Connecting to:          mongodb://127.0.0.1:27018/?directConnection=true&serverS
Using MongoDB:          8.0.5
Using Mongosh:          2.4.2

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-03-20T20:37:53.779+07:00: Access control is not enabled for the databa
------

test> use bank
switched to db bank
bank> db.accounts.insertOne({_id: "account_2", balance: 50})
{ acknowledged: true, insertedId: 'account_2' }
```

```
bank> db.accounts.find()
[ { _id: 'account_2', balance: 50 } ]
bank> db.accounts.find()
[ { _id: 'account_2', balance: 80 } ]
bank>
```

Những Điểm Chưa Hoàn Thiện

1. **Rollback chưa có cơ chế retry**
   - Nếu rollback thất bại do lỗi database, hệ thống có thể mất dữ liệu.
   - Cần thêm cơ chế retry hoặc log lỗi vào database để phục hồi thủ công.
2. **Trạng thái giao dịch chưa được lưu trữ bền vững**
   - Hiện trạng thái "prepared" chỉ lưu trong RAM.
   - Nếu server bị restart giữa chừng, dữ liệu giao dịch sẽ mất.
   - Giải pháp: Lưu trạng thái vào MongoDB để có thể khôi phục.
3. **RPC chưa có cơ chế timeout**
   - Nếu một server không phản hồi, Coordinator có thể bị treo.
   - Nên thêm timeout vào RPC call để tránh deadlock.

Tổng Kết

Hệ thống đã triển khai đúng nguyên tắc 2-Phase Commit, đảm bảo tính toàn vẹn dữ liệu. Tuy nhiên, cần bổ sung logging vào MongoDB, cơ chế retry khi rollback, và timeout cho RPC để hệ thống trở nên hoàn thiện hơn.