



Seminararbeit am Institut für Informatik der Freien Universität Berlin,  
Arbeitsgruppe Technische Informatik

## Two-Phase-Commit

Michael Knoch, Yuri Lewash  
Matrikelnummer: 4289388, 4293181

[michael.knoch@inf.fu-berlin.de](mailto:michael.knoch@inf.fu-berlin.de), [yuri.lewash@inf.fu-berlin.de](mailto:yuri.lewash@inf.fu-berlin.de)

Betreuer: Simon Schmidt  
Eingereicht bei: Katinka Wolter

Berlin, 23.07.2015

### **Zusammenfassung**

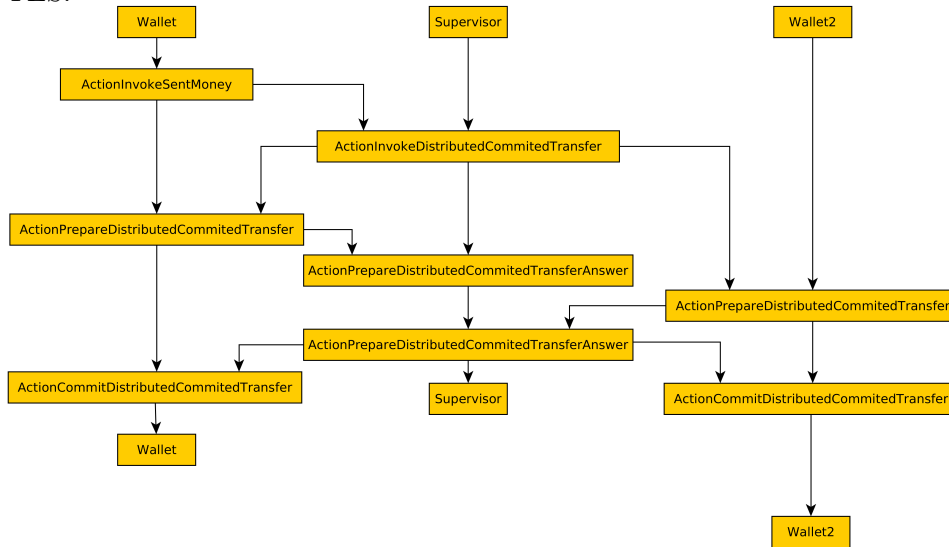
Abstract

## Inhaltsverzeichnis

1	ActionInvokeSentMoney	1
2	ActionInvokeDistributedCommittedTransfer	2
3	ActionPrepareDistributedCommittedTransfer	3
4	ActionPrepareDistributedCommittedTransfer Answer	3
5	ActionCommitDistributedCommittedTransfer	4
6	ActionUpdateQueue	5

## 1 Introduction

We have to make sure, that either all operational participants commit the transaction or none of them. Each participant has one of two votes for the possible transaction: YES or NO. For the transaction to happen all participants have to vote YES, so a decision cannot be reversed. The init participant acts as the coordinator and sends a 'vote request' to all other participants. After a participant receives this request, it responds with either YES or NO. In case of NO the decision is already 'Abort'. If all participants vote YES, the coordinator decides 'Commit' and sends a commit message to all participants. Otherwise it sends an abort message to all participants that voted YES.



## 2 ActionInvokeSentMoney

This action is invoked by the graphical user interface of the wallets. The target is to transfer an amount of money to a given Wallethname. If the targetname has a already a mapping to an ActorRef, an ActionInvokeDistributedCommittedTransfer will be sent to the Supervisor. Otherwise an Gossip is invoked using the ActionSearchWalletReference, and the same ActionInvokeSentMoney is invoked after 200ms.

```

protected void onAction(ActorRef sender,
    ActorRef self,
    UntypedActorContext context,
    Wallet wallet) {
    log(wallet.getKnownNeighbors()+"");
    if(wallet.getKnownNeighbors().containsKey(name)){
        wallet.getRemoteSupervisorActor().tell(
            new ActionInvokeDistributedCommittedTransfer(

```

```

        self,
        wallet.getKnownNeighbors().get(name),
        amount),
        sender);
    }else{
        ActionSearchWalletReference aswr = new
            ActionSearchWalletReference(name);
        for(ActorRef neighbor : wallet.getKnownNeighbors
            ().values()){
            neighbor.tell(aswr, self);
        }
        sleep(self, context, 200);
        self.tell(this, self);
    }
}

```

### 3 ActionInvokeDistributedCommittedTransfer

The ActionInvokeDistributedCommittedTransfer creates an DistributedCommittedTransferRequest, which contains a random generated id, on the Server, with a timeout of 500ms, and store this to a map Long -> Request according to the id. The Timeout is handled by the ActionUpdateQueue explained later. The request will spread out to all clients. The clients can answer with an acknowledgement or an abort afterwards(see below).

```

protected void onAction(ActorRef sender,
                        ActorRef self,
                        UntypedActorContext context,
                        Supervisor supervisor) {
    log("invoke transaction "+source.path().name()+
        " sends "+amount+" to "+target.path().name());
    long timeout = System.currentTimeMillis()+500;
    DistributedCommittedTransferRequest ds =
    new DistributedCommittedTransferRequest(source,target
        ,timeout);
    supervisor.addDistributedCommittedTransferRequest(ds)
        ;
    ActionPrepareDistributedCommittedTransfer apdct =
    new ActionPrepareDistributedCommittedTransfer(
        source,
        target,
        amount,
        timeout,
        ds.getId());
    for(ActorRef neighbor : supervisor.getKnownNeighbors
        ().values()){
        neighbor.tell(apdct, self);
    }
}

```

```
}

```

## 4 ActionPrepareDistributedCommittedTransfer

The clients will reply an request with an acknowlegment if one of two cases occurs. The first case will appear if the Supervisor(Bank) will send a user some money. Otherwise it would be neccessary that the sender is known by this client and have enough money.

```
protected void onAction(ActorRef sender,
                        ActorRef self,
                        UntypedActorContext context,
                        Wallet wallet) {
    //sender is supervisor(bank) has allways
    money
    boolean granted = sender.compareTo(source)==0
    //sender is unknown, might be valid
    ||(wallet.amounts.containsKey(source)
    //sender have enough money
    &&wallet.amounts.getDefault(source,0)>=
    amount);
    sender.tell(
        new ActionPrepareDistributedCommittedTransferAnswer
        (source,
            target,
            amount,
            timestamp,
            granted,
            id),
        self);
}
```

## 5 ActionPrepareDistributedCommittedTransferAnswer

After an answer of a client reaches the server, the server tries to find the correspong request. If the answer was a acknowledgement, the request will get another positive answers. When the same amount of positive answers equals the count of knownneighbors received on the server, all client will be informed to commit the change, and the request will be deleted. If the answer was a Abort all client will be invoked to abort the transaction.

```
protected void onAction(ActorRef sender, ActorRef self
,
    UntypedActorContext context, Supervisor
    superVisor) {
    log(""+superVisor.getKnownNeighbors());
}
```

```

log("granted?" + granted);
DistributedCommittedTransferRequest request =
    superVisor.getRequest(id);
if(granted){
    if(request == null) // unknown
        DistributedCommittedTransferRequest ignore
        return;
    int newCount = request.addPositiveAnswer(sender)
    ;
    if(newCount == superVisor.getKnownNeighbors().
        size()){
        ActionCommitDistributedCommittedTransfer acdct
        = new
            ActionCommitDistributedCommittedTransfer(
                source, target, amount, true, timestamp, id);
        for(ActorRef neighbor : request.getAnswers()){
            neighbor.tell(acdct, self);
        }
        superVisor.deleteRequest(request);
    }
} else {
    // A client wants to rollback
    if(request != null){
        ActionCommitDistributedCommittedTransfer acdct
        = new
            ActionCommitDistributedCommittedTransfer(
                source, target, amount, false, timestamp, id);
        for(ActorRef neighbor : request.getAnswers()){
            neighbor.tell(acdct, self);
        }
    }
}
}
}

```

## 6 ActionCommitDistributedCommittedTransfer

If a client shall commit the given changes, it will perform the changes on the amounts map. Otherwise it will just print a abort transaction message.

```

protected void onAction(ActorRef sender, ActorRef self
,
    UntypedActorContext context, Wallet wallet) {
    log("ActionCommitDistributedCommittedTransfer is
        granted?" + granted);
    if(granted){
        Integer sourceAmount = wallet.amounts.
            getOrDefault(source, 0);
    }
}

```

```

        Integer targetAmount = wallet.amounts.
            getOrDefault(target,0);
        wallet.amounts.put(source,sourceAmount-amount);
        wallet.amounts.put(target,targetAmount+amount);
        if(source.compareTo(self)==0)wallet.amount -=
            amount;
        else if(target.compareTo(self)==0)wallet.amount
            +=amount;
        wallet.log("have now "+wallet.amounts.get(self)+
            " Fucoins");
    }else{
        log("abort transaction with id"+id);
    }
    log("wallet.amounts:"+wallet.amounts);
}

```

## 7 ActionUpdateQueue

the ActionUpdateQueue Event will be invoked each second on the server, and removes all outdated request. If a request is deleted all clients, will be informed to abort this transaction.

```

protected void onAction(ActorRef sender, ActorRef self
    ,
    UntypedActorContext context, Supervisor
        supervisor) {

    List<DistributedCommittedTransferRequest> deletes =
        supervisor.updateList();

    for(DistributedCommittedTransferRequest
        outdatedRequest : deletes){
        ActionCommitDistributedCommittedTransfer acdct =
            new ActionCommitDistributedCommittedTransfer(
                outdatedRequest);
        for(ActorRef neighbor : supervisor.
            getKnownNeighbors().values()){
            neighbor.tell(acdct, self);
        }
    }
    sleep(self,context,1000);
    self.tell(this, self);
}

```