Distributed Applications
Shared data and distributed transactions

BS Doherty

`b.s.doherty@aston.ac.uk`

Aston University

Introduction

There are two conflicting requirements:

- Client operations must be prevented from interfering with one another
- Clients should be able to use servers to share and exchange information

Preventing interference

- Single thread operation
- Atomic operations at server
- Synchronisation of server operations in client transactions

Client-server interaction

- Software support as client and server need to converse
- *open* and *close conversation*
- Stateful servers

Distributed Transactions

- Distributed and nested transactions .
- *Simple* distributed transactions
- Co-ordination of distributed transactions

Transactions

- *atomic*
- synchronize changes of state so that they all occur, or none occur.

Atomicity

There are two aspects of atomicity:

- all-or-nothing - a transaction either completes correctly and alters system state or it fails and does not alter system state. There are two subgroups:
  - failure atomicity:
  - durability:
- isolation

For failure atomicity and durability, data items must be *recoverable*

Transactional service

- A *transactional service* ... waiting for a *closeOperation* signal from the client
- At this stage the server performs a *commit* operation, and returns a *commit* signal.
- If the transaction has not proceeded normally an *abort* signal is returned.
- Abortion of a transaction can be initiated either by client or server.
- Both should be capable of recovery from failure, either their own or the other party, preferably without human intervention.

one-phase commitment protocol

- In a distributed transaction, the client has requested operations at more than one server.
- A transaction comes to an end when the client requires that a transaction should be committed or aborted.
- A simple way to complete the transaction in an atomic manner is for the coordinator to communicate the commit or abort request to all of the servers in the transaction and keep on repeating the request until all of them have acknowledged that they have carried it out.

“ACID” properties

- **Atomicity** All the operations of a transaction must take place, or none of them do
- **Consistency** The completion of a transaction must leave the participants in a “consistent” state, whatever that means. For example, the number of owners of a resource must remain at one
- **Isolation** The activities of one transaction must not affect any other transactions
- **Durability** The results of a transaction must be persistent

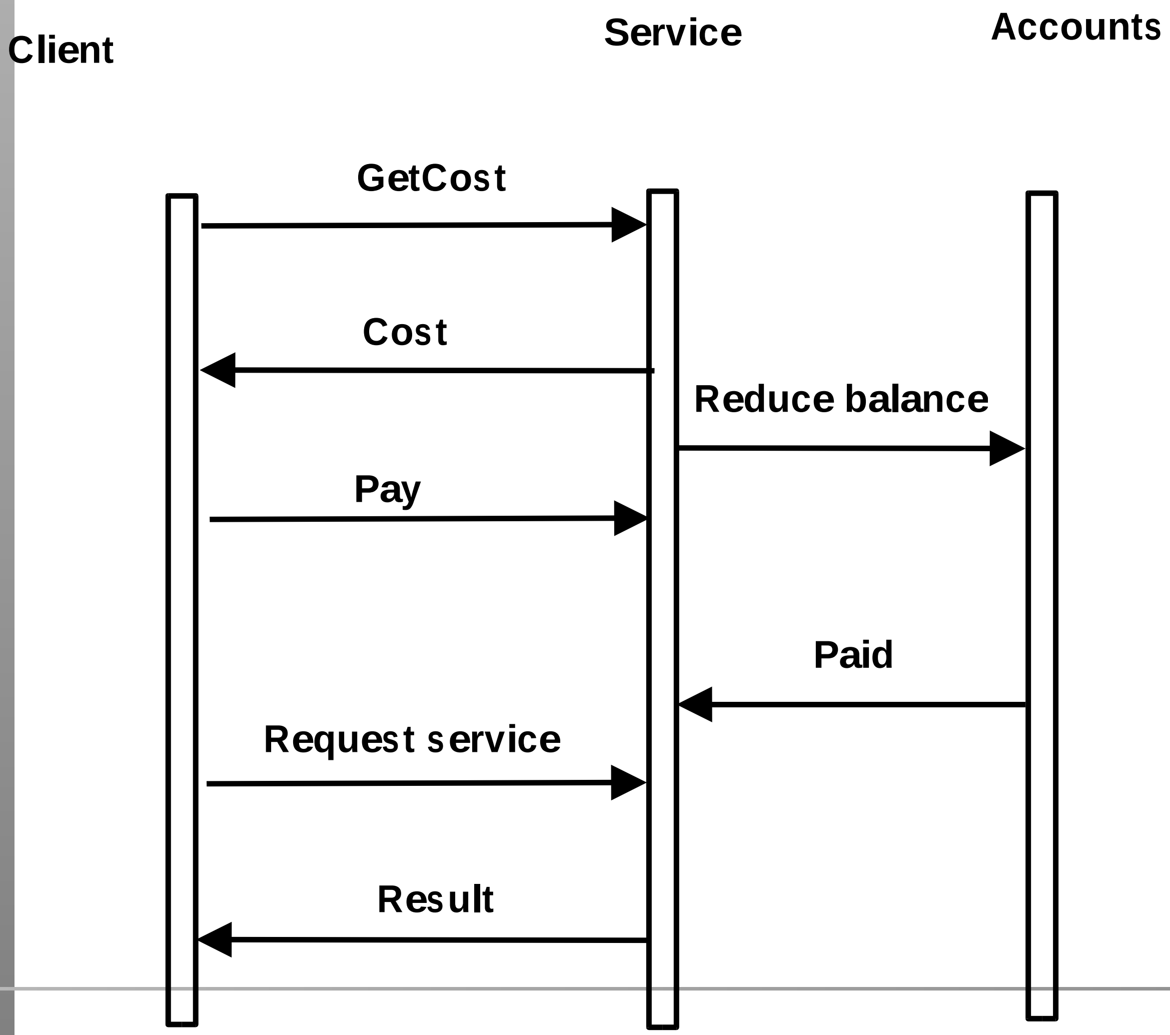
two-phase commit protocol

.
This requires that participants in a transaction be asked to “vote” on a transaction.
If all agree to go ahead, then the transaction “commits”, which is binding on all the participants.
If any “abort” during this voting stage then it forces abortion of the transaction on all participants.

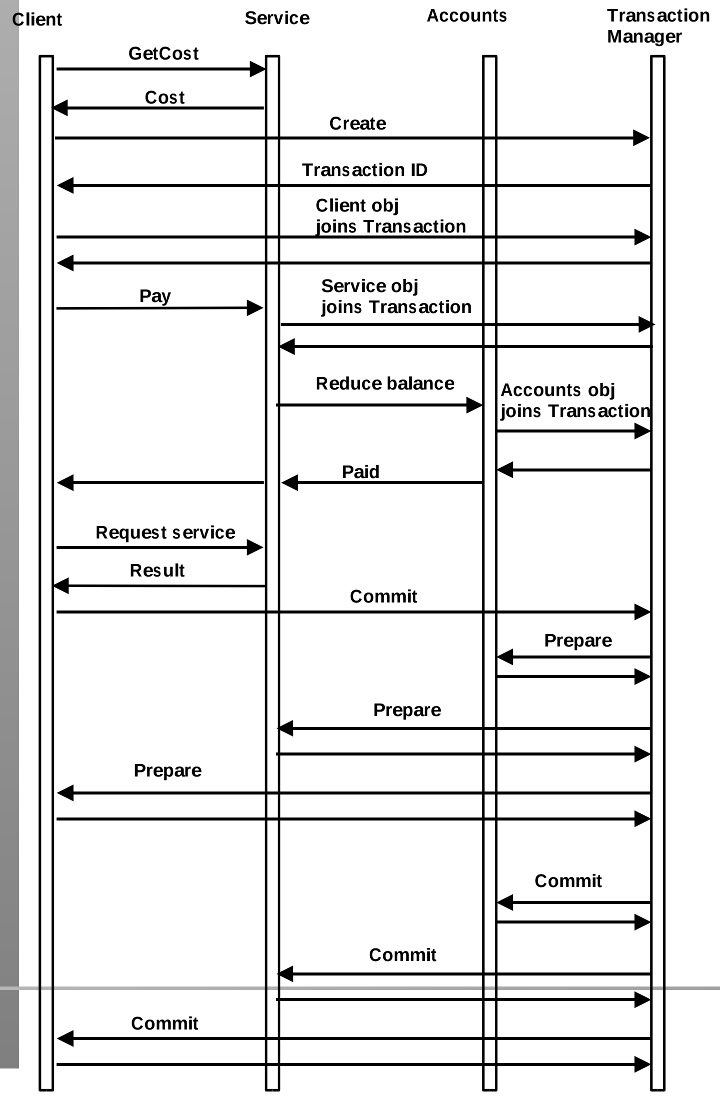
Example

A service may decide to charge for its use.
If a client decides cost is reasonable, it will first credit the service and then request that the service be performed.
The actual accounts will be managed by an accounts service, which will need to be informed of the credits and debits that occur.
A simple accounts model is that the service gets, say, a customer ID from the client, and passes its own ID and the customer ID to the accounts service which manages both accounts.
Similar to the way credit cards work!

Simple transaction



Two-phase commit



Points of failure in transaction

- The cost may be too high for the client.
- The client may offer too little by way of payment to the service.
- There may be a time delay between finding the price and asking for the service.
- After the service is performed, the client may decide that the result was not good enough, and refuse to pay.
- The accounts service may abort the transaction if sufficient client funds are unavailable

Concurrency control

[Coulouris et al., 2001]
The lost update problem: Consider that case where there are three bank accounts, X,Y,Z, initially with balances of £100, £200, £300 respectively. Transaction T1 transfers £4 from X to Y making the balances of X and Y £96 and £204 respectively. Transaction T2 transfers £3 from Z to Y, leaving the balances of Y as £207 and the balance of Z as £297. The net effect is to increase the balance of Y by 7.

Concurrent running

If however both T1 and T2 run concurrently, they both read the balance of Y as £200, add the amount they are transferring to £200, and write back the new balance - which will be £204 or £203, depending on which writes later.

Inconsistent retrievals

occur if for example a bank balance is read by a process T1 in the middle of a processing sequence T2 which is altering the balance.

Recoverability

[Coulouris et al., 2001]
The effects of all committed transactions must be recorded, and none of the effects of aborted transactions should be recorded.

- Dirty reads
- Recoverability of transactions
- Cascading aborts
- Premature writes
- Strict execution of transactions

Fault tolerance and recovery

- *stateless* server [Coulouris et al., 2001].
- Recoverable data items

TP monitors

Transaction processing monitors control the transaction process, managing concurrent execution of threads and processes involved in the transaction.

TP monitors also ensure that the ACID properties are met. [Ince, 2004]

Enterprise Java Beans

Ince [Ince, 2004]gives an example of implementation of a TP monitor using Enterprise Java Beans technology.

References

References

[Coulouris et al., 2001] Coulouris, G., Dollimore, J., and Kindberg, T. (2001). *Distributed Systems Concepts and Design*. Addison-Wesley, third edition.

[Ince, 2004] Ince, D. (2004). *Developing Distributed and E-commerce Applications*. Addison Wesley, second edition.