Announcements

• Adrian office hours: Tu 2:30-4:30
• Both texts should be on reserve by now
• Short lecture Thurs 11:40-12:30
  • Devoted to project discussion
This Lecture

- **RPC Systems**
  - [BN97] Ch 3
  - [ACKM04] Sec 2.2

- **TP Monitors**
  - [BN97] Ch 2,
  - [ACKM04] Sec 2.3
  - [BN97] Ch 9 (2PC)
Types of Middleware

- RPC-based systems
- TP Monitors
- Object Brokers
- Object Monitors
- Message-Oriented Middleware (MOM)
- Message Brokers
RPC Systems

• The foundation for most of the others
• We’ll cover in some depth
• Requires support infrastructure
  • Interface Definition Language (IDL) compilers, stub generators
  • directory services for binding
  • etc.
TP Monitors

- Most established form of middleware
- TP-Lite
  - A 2-Tier architecture
  - Allow application logic as stored procedures in a database
- TP-Heavy
  - 3-Tier architecture
  - Implements transactional RPC
  - Coordinator for distributed transactions
Objects

- Object Broker (OMG CORBA)
  - Like an object-oriented RPC system
- Object Monitor
  - Like an object-oriented TP Monitor
Queues

- Message-Oriented Middleware
  - Queues (and transactional queues) to support asynchronous messaging

- Message Broker
  - All of the above
  - Ability to run application logic for message routing
Remote Procedure Call: hides communication details behind a procedure call and helps bridge heterogeneous platforms.

Sockets: operating system level interface to the underlying communication protocols.

TCP, UDP:
- User Datagram Protocol (UDP) transports data packets without guarantees.
- Transmission Control Protocol (TCP) verifies correct delivery of data streams.

Internet Protocol (IP): moves a packet of data from one node to another.
RPC Development

- Interface Description Language (IDL) in which signatures of procedures are described
- IDL Compiler generates caller & callee stubs
- Stubs are responsible for marshalling, transmitting and unmarshalling arguments and results
RPC Development

client process

client code

language specific call interface

client stub

development environment

IDL

IDL sources

IDL compiler

interface headers

server process

server code

language specific call interface

server stub
Basic RPC Runtime

Client process:
- Procedure call
- Client stub
  - Bind
  - Marshal
  - Serialize
  - Send
- Communication module

Server process:
- Procedure
- Server stub
  - Unmarshal
  - Deserialize
  - Receive
- Dispatcher (select stub)
- Communication module
Naming & Directory Service

- Callee registers with NDS
- Caller looks up callee by name & signature
- Possibly multiple matches
  - => traders
- Possibly multiple server instances
  - => potential for load balancing
- Possibly no active server instances
  - => start one?
Dynamic Binding for RPC

1. register server and procedure
2. find
3. query for server implementing the procedure
4. address of server
5. send
6. invoke procedure
7. receive

client stub
bind
marshal
serialize

3. query for server implementing the procedure

4. address of server

name and directory service (binder)

client process

communication module

server process

communication module

dispatcher (select stub)

server stub
0. register
unmarshal
deserialize

1. register
server and procedure
Parameter Translation

• Canonical encoding on wire
  • the $n^2$ problem

• Receiver-translates
  • best performance if homogeneous
Security

- Advantageous to build authentication into RPC infrastructure
- Discussion deferred until later
Fault Tolerance

- Retries
  - Reliable transport?
  - If idempotent => seek *at least once*
  - Not idempotent => *at most once*
- See *Transactional RPC*
RPC Performance

- Procedure invocation overhead 100-1000 times greater for RPC than local call
- *Plus* the communication latency
- 15,000 machine instructions don’t take very long these days ...
DCE Environment

- Client process
  - Client code
  - Language specific call interface
    - Client stub
  - RPC API
    - RPC run time service library

- DCE development environment
  - IDL
  - IDL sources
  - IDL compiler
    - Interface headers

- Server process
  - Server code
  - Language specific call interface
    - Server stub
  - RPC API
    - RPC run time service library

- RPC protocols
- Security service
- Cell service
- Distributed file service
- Thread service

DCE runtime environment
Suppose RPCs done in application that requires transactional ACID properties

Distribution makes this difficult ...
Distributed Commit

- RM1 and RM2 must both commit or both abort
- What if one of them fails during commit protocol?
- Failure model is not Byzantine or FailStop but Crash-Recover
Solution: TP Monitor

- Transaction Manager
  - Multiple RPCs between BOT-EOT calls execute as one transaction
- Coordinator for distributed 2-Phase Commit
  - (Details of 2PC later ... )
Transactional RPC

**Client Process**
1. BOT
4. procedure call
10. EOT
2. register txn & create context
5. add txn id & context to call
11. request commit
3. create txn id
6. extract context and txn id
7. register server for txn
13. participate in 2PC
8. lookup txn id
12. lookup txn id
9. procedure

**Server Process**

**Client Stub**
- register txn & create context
- add txn id & context to call
- request commit
- confirm termination

**Server Stub**
- extract context and txn id
- register server for txn
- participate in 2PC
- lookup txn id

**Transaction Manager**
- create txn id
- register txn
- register client for txn
- return txn id
- lookup txn id
- register server for txn
- run 2PC
- notify client of outcome
If there is more than one transaction server, the commit coordinator has to be in the upper tier ...
The 3 Tiers match the 3 application layers ...
3-Tier TP Monitor ...

- 3-Tier Model Maps to Object-Oriented Application ...

Presentation Server

Workflow Controller

Transaction Server

“Open Account” Menu Item

Open an Account

Customer Object

Account Object

3-Tier TP Monitor

Object-Oriented Application Architecture
2-Tier system requires quadratically many edges (sessions)
3-Tier Communication

Presentation Server ➔ Transaction Server ➔ Transaction Server ➔ Presentation Server

Presentation Server ➔ Transaction Server ➔ . . . ➔ Transaction Server ➔ Presentation Server

Presentation Server ➔ Transaction Server ➔ Transaction Server ➔ Presentation Server

Presentation Server ➔ Transaction Server ➔ Transaction Server ➔ Presentation Server

3-Tier system requires only linearly many sessions
2-Phase Commit

- Phase 1:
  - Coordinator sends PREPARE to all participants and waits for responses
  - Participants reply YES or NO, or fail to reply
2-Phase Commit

- Phase 2:
  - Coordinator decides YES iff received YES votes from all participants
  - Coordinator sends decision to all participants
  - Participants reply DONE
  - Coordinator frees resources after receiving DONE from all participants
2-Phase Commit - Blocking

- **Correctness:**
  - After voting NO participant may abort
  - After voting YES participant may not commit or abort until receiving the coordinator decision -- *in doubt*

- What if coordinator fails while some participants are in doubt? *Blocked!*
2-Phase Commit - Theorems

- For every possible distributed commit protocol, a communication failure can cause a participant to become blocked.

- No distributed commit protocol can guarantee independent recovery (recovery without cooperation from coordinator) of failed participants.
Logging in 2PC

- Coordinator and participants must log enough information to enable recovery if a failure occurs during execution of the 2PC protocol
Logging in 2PC

1. Log a START record
2. Log a PREPARE record
3. Log a COMMIT record
4. Log a COMMITTED record
5. Log a DONE record
Logging in 2PC

1. Log a START record

2. Log a NO record

3. Log an ABORT record

4. Log an ABORTED record

5. Log a DONE record