Advanced Distributed Systems

Course Plan and Overview

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

Outline

Lectures - I

Lecture	Topic
1	Introduction and Structure
2	Communication – Client Server, RPC
	Message Queue, Stream
3	Multicast Communication – Epidemic, Gossip
4	Naming : DNS, Chord
5	Naming : Pastry, Tapestry, LDAP
6	Physical Clock , Logical Clock, Totally ordered multicast
7	Mutual Exclusion Algorithms
8	Leader Election Algorithms
9	Consistency - I
10	Consistency - II
11	Fault Tolerance – Introduction, Log based recovery
12	Byzantine Fault Tolerance

Lectures - II

Lecture	Topic
13	Paxos
14	Distributed Commit
15	Security – Channels, Concepts
16	Security Applications – Kerberos
	Diffie Helmann Key Exchange
17	Corba, EJB
18	AFS and NFS
19	Akamai and Corona
20	Dynamo and Voldemort
21	Coda, Fawn, and Google FS
22	Web Services
23	Dryad-LinQ

Grading Scheme

Component	Weightage
Attendance	10%
Midterm	15%
End term (take home)	25%
Programming Assignment 1	15%
Programming Assignment 2	15%
Programming Assignment 3	20%

Outline

Distributed System: Definition

A distributed system is a piece of software that ensures that:

> *a collection of independent computers appears to its users as a single coherent system*

Two aspects:

1. independent computers
2. single coherent system $\Rightarrow$ middleware .

Goals of Distributed Systems

Goals

- Making resources available
- Distribution transparency
- Openness
- Scalability

Distribution Transparency

Transp.	Description
Access	Hides differences in data representation and invocation mechanisms
Location	Hides where an object resides
Migration	Hides from an object the ability of a system to change that object's location
Relocation	Hides from a client the ability of a system to change the location of an object to which the client is bound
Replication	Hides the fact that an object or its state may be replicated and that replicas reside at different locations
Concurrency	Hides the coordination of activities between objects to achieve consistency at a higher level
Failure	Hides failure and possible recovery of objects

Distribution Transparency

Transp.	Description
Access	Hides differences in data representation and invocation mechanisms
Location	Hides where an object resides
Migration	Hides from an object the ability of a system to change that object's location
Relocation	Hides from a client the ability of a system to change the location of an object to which the client is bound
Replication	Hides the fact that an object or its state may be replicated and that replicas reside at different locations
Concurrency	Hides the coordination of activities between objects to achieve consistency at a higher level
Failure	Hides failure and possible recovery of objects

Note

Distribution transparency is a nice a goal, but achieving it is a different story.

Degree of Transparency

- Aiming at full distribution transparency may be too much

Degree of Transparency

- Aiming at full distribution transparency may be too much
- Users may be located in different continents

Degree of Transparency

- Aiming at full distribution transparency may be too much
- Users may be located in different continents
- Completely hiding failures of networks and nodes is (theoretically and practically) impossible
  - You cannot distinguish a slow computer from a failing one
  - You can never be sure that a server actually performed an operation before a crash

Degree of Transparency

- Aiming at full distribution transparency may be too much
- Users may be located in different continents
- Completely hiding failures of networks and nodes is (theoretically and practically) impossible
  - You cannot distinguish a slow computer from a failing one
  - You can never be sure that a server actually performed an operation before a crash
- Full transparency will cost performance , exposing distribution of the system
  - Keeping Web caches exactly up-to-date with the master
  - Immediately flushing write operations to disk for fault tolerance

Openness of Distributed Systems

- Open distributed system $\Rightarrow$ Be able to interact with services from other open systems, irrespective of the underlying environment:
  - Systems should conform to well-defined interfaces
  - Systems should support portability of applications
  - Systems should easily interoperate

Openness of Distributed Systems

- Open distributed system $\Rightarrow$ Be able to interact with services from other open systems, irrespective of the underlying environment:
  - Systems should conform to well-defined interfaces
  - Systems should support portability of applications
  - Systems should easily interoperate
- Achieving openness $\Rightarrow$ At least make the distributed system independent from heterogeneity of the underlying environment:
  - Hardware
  - Platforms
  - Languages

Scale in Distributed Systems

Observation

Many developers of modern distributed system easily use the adjective "scalable" without making clear **why** their system actually scales.

Scale in Distributed Systems

Observation

Many developers of modern distributed system easily use the adjective "scalable" without making clear **why** their system actually scales.

Scalability – At least three components:

- Number of users and/or processes (size scalability)
- Maximum distance between nodes (geographical scalability)
- Number of administrative domains (administrative scalability)

Scale in Distributed Systems

Observation

Many developers of modern distributed system easily use the adjective “scalable” without making clear **why** their system actually scales.

Scalability – At least three components:

- Number of users and/or processes (size scalability)
- Maximum distance between nodes (geographical scalability)
- Number of administrative domains (administrative scalability)

Observation

Most systems account only, to a certain extent, for size scalability. The (non)solution: powerful servers. Today, the challenge lies in geographical and administrative scalability.

Techniques for Scaling

Hide communication latencies – Avoid waiting for responses; do something else:

- Make use of asynchronous communication
- Have separate handler for incoming response
- Problem: not every application fits this model

Techniques for Scaling

Distribution

Partition data and computations across multiple machines:

- Move computations to clients (Java applets)
- Decentralized naming services (DNS)
- Decentralized information systems (WWW)

Techniques for Scaling

Replication/caching

Make copies of data available at different machines:

- Replicated file servers and databases
- Mirrored Web sites
- Web caches (in browsers and proxies)
- File caching (at server and client)

Scaling – The Problem

Observation

Applying scaling techniques is easy, except for one thing:

Scaling – The Problem

Observation

Applying scaling techniques is easy, except for one thing:

- Having multiple copies (cached or replicated), leads to inconsistencies : modifying one copy makes that copy different from the rest.

Scaling – The Problem

Observation

Applying scaling techniques is easy, except for one thing:

- Having multiple copies (cached or replicated), leads to inconsistencies : modifying one copy makes that copy different from the rest.
- Always keeping copies consistent and in a general way requires global synchronization on each modification.

Scaling – The Problem

Observation

Applying scaling techniques is easy, except for one thing:

- Having multiple copies (cached or replicated), leads to inconsistencies : modifying one copy makes that copy different from the rest.
- Always keeping copies consistent and in a general way requires global synchronization on each modification.
- Global synchronization precludes large-scale solutions.

Scaling – The Problem

Observation

Applying scaling techniques is easy, except for one thing:

- Having multiple copies (cached or replicated), leads to inconsistencies : modifying one copy makes that copy different from the rest.
- Always keeping copies consistent and in a general way requires global synchronization on each modification.
- Global synchronization precludes large-scale solutions.

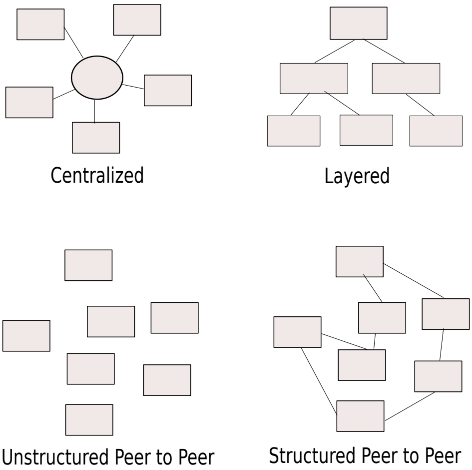
Observation

If we can tolerate inconsistencies, we may reduce the need for global synchronization, but tolerating inconsistencies is application dependent .

Outline

1. Plan
2. Introduction
   - Overview
   - Developing Distributed Systems: Pitfalls
3. Building a Distributed System
   - Computation
   - Communication
   - Remote Procedure Calls
   - Message-Oriented Communication

Types of Distributed Systems : Structure



Types of Distributed Systems : Synchronous vs Asynchronous

Definition

Synchronous System The requester waits for the response before placing other requests.

Definition

Asynchronous System The requester does not wait for the response before placing other requests. One example of such systems are publish/subscribe systems. A given set of nodes subscribe for a service. When a node is ready to publish some data, it looks up the list of subscribers and sends them messages.

Developing Distributed Systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. There are many false assumptions :

Developing Distributed Systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. There are many false assumptions :

- The network is reliable

Developing Distributed Systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. There are many false assumptions :

- The network is reliable
- The network is secure

Developing Distributed Systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. There are many false assumptions :

- The network is reliable
- The network is secure
- The network is homogeneous

Developing Distributed Systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. There are many false assumptions :

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change

Developing Distributed Systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. There are many false assumptions :

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero

Developing Distributed Systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. There are many false assumptions :

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite

Developing Distributed Systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. There are many false assumptions :

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero

Developing Distributed Systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. There are many false assumptions :

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is one administrator

Computation and Communication

Components of a distributed System

- Nodes that run the distributed software. (Computation)
- Communication network. (Communication)

Outline

Computing Nodes

- A compute node can either be a process or a thread

Thread

A thread is a light weight process that shares the address space with other threads.

Process

A process is the runtime image of a program. It does not share its address space.

- We can use regular memory reads and writes to communicate across threads. However, inter-process communication requires OS intervention .
- In practice, there are thousands of threads and processes distributed across hundreds of sites.

Computing Nodes - II

- Generic solution: Message passing across threads/processes.
- Minimize shared data.
- Fetch shared data through messages from a database.

Outline

How do Nodes Talk to Each Other?

- Distributed systems predominantly use application layer protocols.
- This protocol layer is known as middleware .
- They use standard sockets to send messages to other nodes.
- Paradigms for sending messages
  1. Synchronous vs Asynchronous
  2. Transient vs Persistent

Middleware Layer

Observation

Middleware is invented to provide **common** services and protocols that can be used by many different applications

- A rich set of communication protocols
- (Un)marshaling of data, necessary for integrated systems
- Naming protocols , to allow easy sharing of resources
- Security protocols for secure communication
- Scaling mechanisms , such as for replication and caching

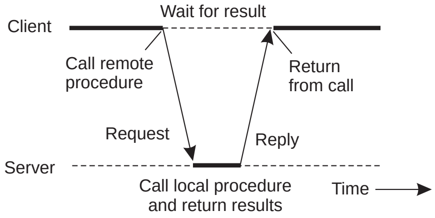
Note

What remains are truly application-specific protocols...
such as?

Outline

Basic RPC operation

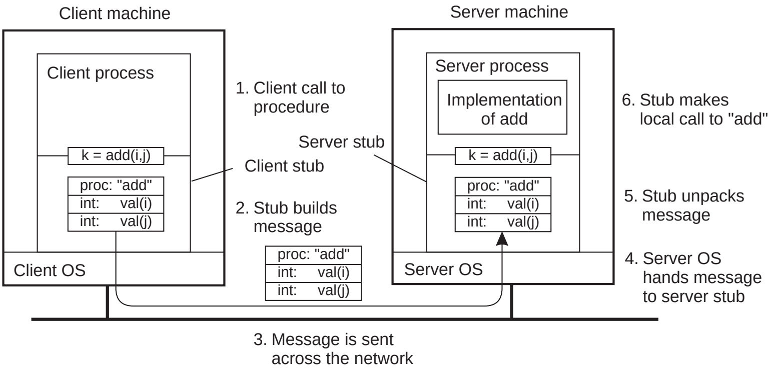
- Application developers are familiar with a simple procedure model
- Well-engineered procedures operate in isolation (black box)
- There is no fundamental reason not to execute procedures on separate machines



Conclusion

Communication between caller & callee can be hidden by using procedure-call mechanism.

Basic RPC


Client machine
Server machine
Client process
Server process
Implementation of add
1. Client call to procedure
6. Stub makes local call to "add"
Server stub
k = add(i,j)
k = add(i,j)
Client stub
proc: "add"
int: val(i)
int: val(j)
proc: "add"
int: val(i)
int: val(j)
5. Stub unpacks message
2. Stub builds message
proc: "add"
int: val(i)
int: val(j)
4. Server OS hands message to server stub
Client OS
Server OS
3. Message is sent across the network

Basic RPC operation

1. Client procedure calls client stub.
2. Stub builds message; calls local OS.
3. OS sends message to remote OS.
4. Remote OS gives message to stub.
5. Stub unpacks parameters and calls server.

1. Server returns result to stub.
2. Stub builds message; calls OS.
3. OS sends message to client's OS.
4. Client's OS gives message to stub.
5. Client stub unpacks result and returns to the client.

RPC: Parameter passing

Parameter marshaling

There's more than just wrapping parameters into a message:

- Client and server machines may have different data representations (think of byte ordering)
- Wrapping a parameter means transforming a value into a sequence of bytes
- Client and server have to agree on the same encoding :
  - How are basic data values represented (integers, floats, characters)
  - How are complex data values represented (arrays, unions)
- Client and server need to properly interpret messages , transforming them into machine-dependent representations.

RPC: Parameter passing

RPC parameter passing: some assumptions

- Copy in/copy out semantics: while procedure is executed, nothing can be assumed about parameter values.
- All data that is to be operated on is passed by parameters. Excludes passing references to (global) data.

Conclusion

Full access transparency cannot be realized.

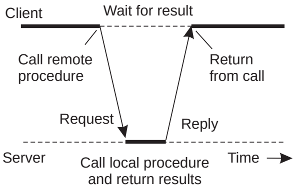
A remote reference mechanism enhances access transparency:

- Remote reference offers unified access to remote data
- Remote references can be passed as parameter in RPCs

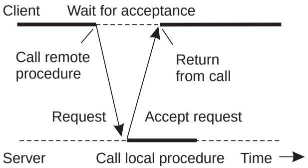
Asynchronous RPCs

Essence

Try to get rid of the strict request-reply behavior, but let the client continue without waiting for an answer from the server.

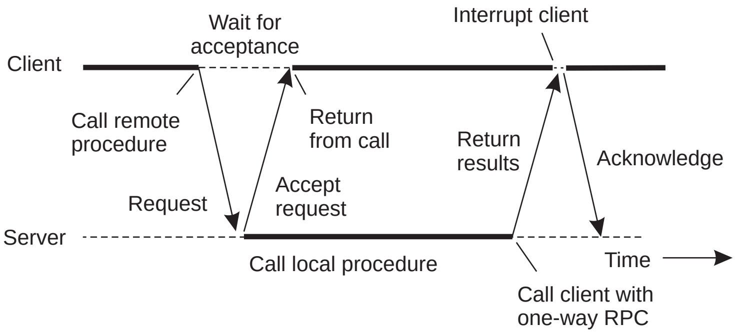


(a)



(b)

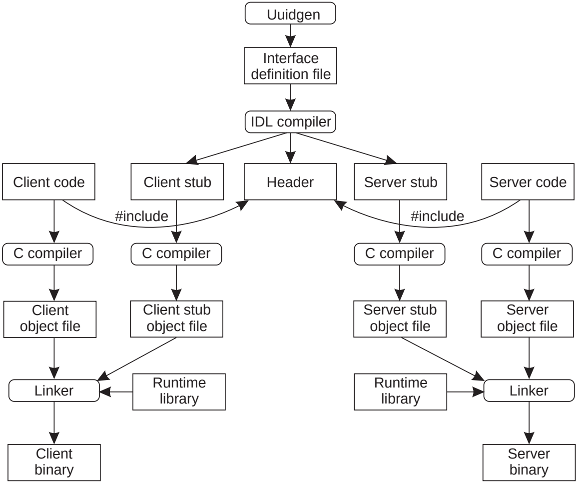
Deferred synchronous RPCs



Variation

Client can also do a (non)blocking poll at the server to see whether results are available.

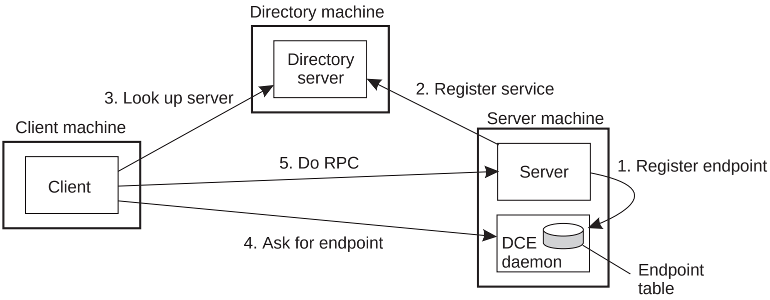
RPC in practice



Client-to-server binding (DCE)

Issues

(1) Client must locate server machine, and (2) locate the server.



Outline

Message-Oriented Communication

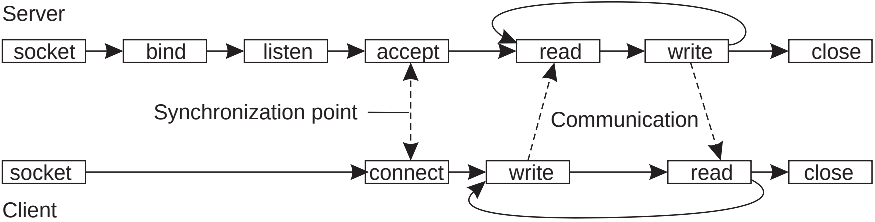
- Transient Messaging
- Message-Queuing System
- Message Brokers
- Example: IBM WebSphere

Transient messaging: sockets

Berkeley socket interface

SOCKET	Create a new communication endpoint
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept N connections
ACCEPT	Block until request to establish a connection
CONNECT	Attempt to establish a connection
SEND	Send data over a connection
RECEIVE	Receive data over a connection
CLOSE	Release the connection

Transient messaging: sockets



Message-oriented middleware

Essence

Asynchronous persistent communication through support of middleware-level queues. Queues correspond to buffers at communication servers.

PUT	Append a message to a specified queue
GET	Block until the specified queue is nonempty, and remove the first message
POLL	Check a specified queue for messages, and remove the first. Never block
NOTIFY	Install a handler to be called when a message is put into the specified queue

Message broker

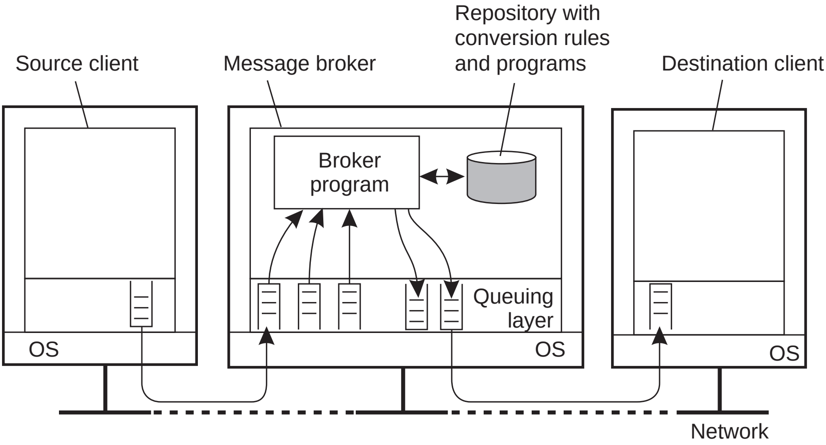
Observation

Message queuing systems assume a common messaging protocol : all applications agree on message format (i.e., structure and data representation)

Message broker: Centralized component that takes care of application heterogeneity in an MQ system

- Transforms incoming messages to target format
- Very often acts as an application gateway
- May provide subject-based routing capabilities ⇒ Enterprise Application Integration

Message broker



IBM's WebSphere MQ

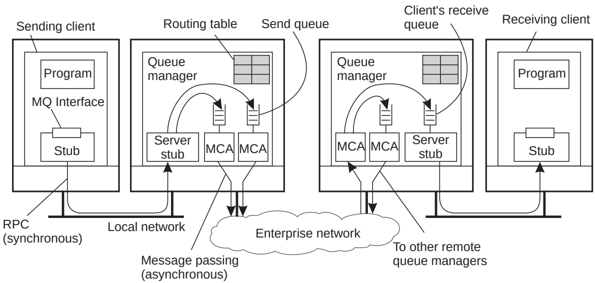
- Application-specific messages are put into, and removed from queues
- Queues reside under the regime of a queue manager
- Processes can put messages only in local queues, or through an RPC mechanism

IBM's WebSphere MQ

Message Transfer

- Messages are transferred between queues
- Message transfer between queues at different processes, requires a channel
- At each endpoint of channel is a message channel agent
- Message channel agents are responsible for:
  - Setting up channels using lower-level network communication facilities (e.g., TCP/IP)
  - (Un)wrapping messages from/in transport-level packets
  - Sending/receiving packets

IBM’s WebSphere MQ



- Channels are inherently unidirectional
- Automatically start MCAs when messages arrive
- Any network of queue managers can be created
- Routes are set up manually (system administration)

IBM's WebSphere MQ

Routing

By using logical names , in combination with name resolution to local queues, it is possible to put a message in a remote queue