IT4371: Distributed Systems

Spring 2016

Communication in Distributed Systems

Dr. Nguyen Binh Minh

Department of Information Systems
School of Information and Communication Technology
Hanoi University of Science and Technology

Today…

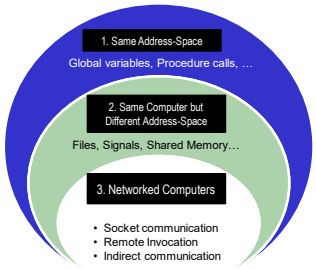
- Last Session:
  - Networking principles
- Today's Session:
  - Communication in Distributed Systems
    - Inter-Process Communication, Remote Invocation, Indirect Communication

Communication Paradigms

Communication paradigms describe and classify a set of methods for the exchange of data between entities in a Distributed System

Classification of Communication Paradigms

Communication Paradigms can be categorized into three types based on where the entities reside. If entities are running on:



Today, we are going to study how entities that reside on networked computers communicate in Distributed Systems

- **Socket communication**
- **Remote Invocation**
- **Indirect communication**

Communication Paradigms

Socket communication
- Low-level API for communication using underlying network protocols

Remote Invocation
- A procedure call abstraction for communicating between entities

Indirect Communication
- Communicating without direct coupling between sender and receiver

Communication Paradigms

Socket communication
Remote invocation
Indirect communication

Socket Communication

Socket is a communication end-point to which an application can write or read data

Socket abstraction is used to send and receive messages from the transport layer of the network

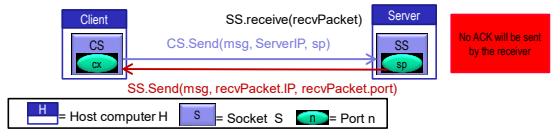
Each socket is associated with a particular type of transport protocol

1. UDP Socket:
   - Provides Connection-less and unreliable communication
2. TCP Socket:
   - Provides Connection-oriented and reliable communication

1. UDP Sockets

Messages are sent from sender process to receiver process using UDP protocol.
- UDP provides connectionless communication, with no acknowledgements or message transmission retries

Communication mechanism:
- Server opens a UDP socket *SS* at a known port *sp*,
- Socket *SS* waits to receive a request
- Client opens a UDP socket *CS* at a random port *cx*
- Client socket *CS* sends a message to ServerIP and port *sp*
- Server socket SS may send back data to *CS*



UDP Sockets – Design Considerations

Messages may be delivered out-of-order
- If necessary, programmer must re-order packets

Communication is not reliable
- Messages might be dropped due to check-sum error or buffer overflows at routers

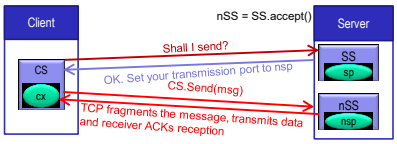
Sender must explicitly fragment a long message into smaller chunks before transmitting
- A maximum size of 548 bytes is suggested for transmission

Receiver should allocate a buffer that is big enough to fit the sender's message
- Otherwise the message will be truncated

2. TCP Sockets

Messages are sent from sender to receiver using TCP protocol
- TCP provides in-order delivery, reliability and congestion control

Communication mechanism
- Server opens a TCP server socket *SS* at a known port *sp*
- Server waits to receive a request (using *accept* call)
- Client opens a TCP socket *CS* at a random port *cx*
- *CS* initiates a connection initiation message to ServerIP and port *sp*
- Server socket SS allocates a new socket NSS on random port *nsp* for the client
- *CS* can send data to *NSS*



Advantages of TCP Sockets

TCP Sockets ensure in-order delivery of messages

Applications can send messages of any size

TCP Sockets ensure reliable communication using acknowledgements and retransmissions

Congestion control of TCP regulates sender rate, and thus prevents network overload

Communication Paradigms

Socket communication

Remote invocation

Indirect communication

Remote Invocation

Remote invocation enables an entity to call a procedure that typically executes on an another computer without the programmer explicitly coding the details of communication

- The underlying middleware will take care of raw-communication
- Programmer can transparently communicate with remote entity

We will study two types of remote invocations:

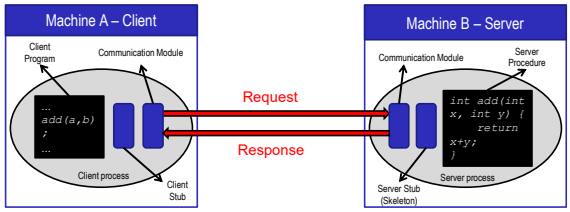
a. Remote Procedure Calls (RPC)
b. Remote Method Invocation (RMI)

Remote Procedure Calls (RPC)

RPC enables a sender to communicate with a receiver using a simple procedure call

- No communication or message-passing is visible to the programmer

Basic RPC Approach



Challenges in RPC

Parameter passing via Marshaling

- Procedure parameters and results have to be transferred over the network as bits

Data representation

- Data representation has to be uniform
  Architecture of the sender and receiver machines may differ

Challenges in RPC

Parameter passing via Marshaling

- Procedure parameters and results have to be transferred over the network as bits

Data representation

- Data representation has to be uniform
  Architecture of the sender and receiver machines may differ

Parameter Passing via Marshaling

Packing parameters into a message that will be transmitted over the network is called parameter marshalling

The parameters to the procedure and the result have to be marshaled before transmitting them over the network

Two types of parameters can passed

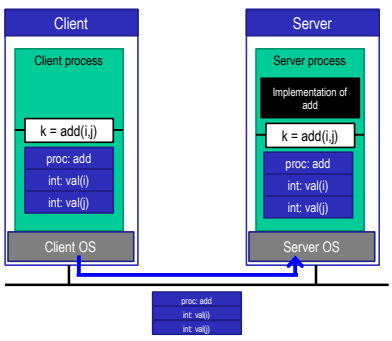
1. Value parameters
2. Reference parameters

1. Passing Value Parameters

Value parameters have complete information about the variable, and can be directly encoded into the message

- e.g., integer, float, character

Example of Passing Value Parameters



2. Passing Reference Parameters

Passing reference parameters like value parameters in RPC leads to incorrect results due to two reasons:

a. Invalidity of reference parameters at the server
Reference parameters are valid only within client's address space
Solution: Pass the reference parameter by copying the data that is referenced

b. Changes to reference parameters are not reflected back at the client
Solution: "Copy/Restore" the data
   - Copy the data that is referenced by the parameter.
   - Copy-back the value at server to the client.

Challenges in RPC

Parameter passing via Marshaling
- Procedure parameters and results have to be transferred over the network as bits

Data representation
- Data representation has to be uniform
  Architecture of the sender and receiver machines may differ

Data Representation

- Computers in DS often have different architectures and operating systems
  The size of the data-type differ
  - e.g., A *long* data-type is 4-bytes in 32-bit Unix, while it is 8-bytes in 64-bit Unix systems

  The format in which the data is stored differ
  - e.g., Intel stores data in little-endian format, while SPARC stores in big-endian format

The client and server have to agree on how simple data is represented in the message
- e.g., format and size of data-types such as integer, char and float

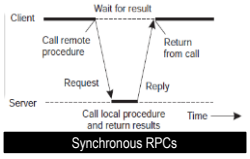
Remote Procedure Call Types

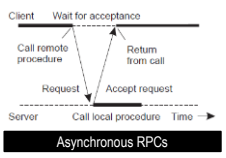
Remote procedure calls can be:
- Synchronous
- Asynchronous (or Deferred Synchronous)

Synchronous vs. Asynchronous RPCs

An RPC with strict request-reply blocks the client until the server returns
- Blocking wastes resources at the client

Asynchronous RPCs are used if the client does not need the result from server
- The server immediately sends an ACK back to client
- The client continues the execution after an ACK from the server



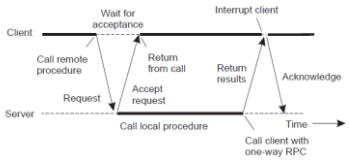


Deferred Synchronous RPCs

Asynchronous RPC is also useful when a client wants the results, but does not want to be blocked until the call finishes

Client uses deferred synchronous RPCs

- Single request-response RPC is split into two RPCs
- First, client triggers an asynchronous RPC on server
- Second, on completion, server calls-back client to deliver the results



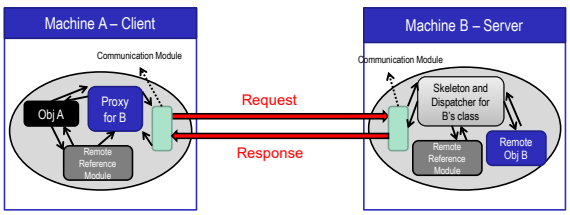
Remote Method Invocation (RMI)

In RMI, a calling object can invoke a method on a potentially remote object

RMI is similar to RPC, but in a world of distributed objects

- The programmer can use the full expressive power of object-oriented programming
- RMI not only allows to pass value parameters, but also pass object references

RMI Control Flow



Communication Paradigms

Socket communication

Remote invocation

Indirect communication

Indirect Communication

Recall: Indirect communication uses middleware to

- Provide one-to-many communication
- Mechanisms eliminate space and time coupling

  Space coupling: Sender and receiver should know each other's identities

  Time coupling: Sender and receiver should be explicitly listening to each other during communication

Approach used: Indirection

- Sender → A Middle-Man → Receiver

Middleware for Indirect Communication

Indirect communication can be achieved through:

1. Message-Queuing Systems
2. Group Communication Systems

Middleware for Indirect Communication

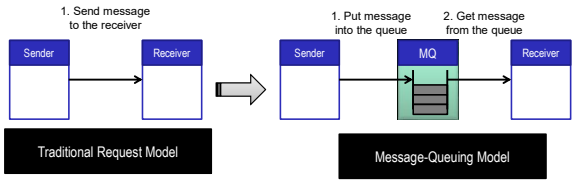
Indirect communication can be achieved through:

1. Message-Queuing Systems
2. Group Communication Systems

Message-Queuing (MQ) Systems

Message Queuing (MQ) systems provide space and time decoupling between sender and receiver

- They provide intermediate-term storage capacity for messages (in the form of Queues), without requiring sender or receiver to be active during communication



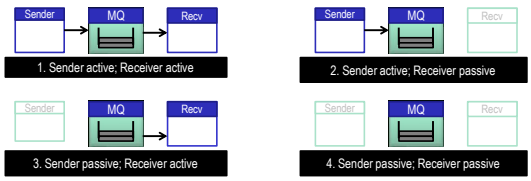
Space and Time Decoupling

MQ enables space and time decoupling between sender and receivers
- Sender and receiver can be *passive* during communication

However, MQ has other types of coupling
- Sender and receiver have to know the identity of the queue
- The middleware (queue) should be always *active*

Space and Time Decoupling (cont'd)

Four combination of loosely-coupled communications are possible in MQ:



Interfaces Provided by the MQ System

Message Queues enable asynchronous communication by providing the following primitives to the applications:

Primitive	Meaning
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 (call-back function) to be called when a message is put into the specified queue

Architecture of an MQ System

The architecture of an MQ system has to address the following challenges:

a. Placement of the Queue
   Is the queue placed near to the sender or receiver?

b. Identity of the Queue
   How can sender and receiver identify the queue location?

c. Intermediate Queue Managers
   Can MQ be scaled to a large-scale distributed system?

a. Placement of the Queue

Each application has a specific pattern of inserting and receiving the messages

MQ system is optimized by placing the queue at a location that improves performance

Typically, a queue is placed in one of the two locations
- Source queues: Queue is placed near the source
- Destination queues: Queue is placed near the destination

Examples:
- "Email Messages" is optimized by the use of destination queues
- "RSS Feeds" requires source queuing

b. Identity of the Queue

In MQ systems, queues are generally addressed by names

However, the sender and the receiver should be aware of the network location of the queue

A naming service for queues is necessary
- Database of queue names to network locations is maintained
- Database can be distributed (similar to DNS)

c. Intermediate Queue Managers

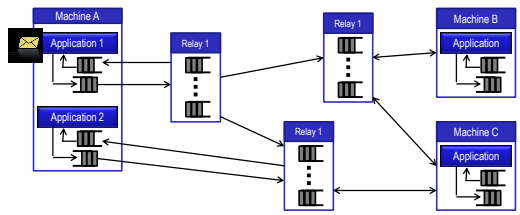
Queues are managed by Queue Managers
- Queue Managers directly interact with sending and receiving processes

However, Queue Managers are not scalable in dynamic large-scale Distributed Systems (DSs)
- Computers participating in a DS may change (thus changing the topology of the DS)
- There is no general naming service available to dynamically map queue names to network locations

Solution: To build an overlay network (e.g., Relays)

c. Intermediate Queue Managers (Cont'd)

Relay queue managers (or relays) assist in building dynamic scalable MQ systems
- Relays act as "routers" for routing the messages from sender to the queue manager



Middleware for Indirect Communication

Indirect communication can be achieved through:

1. Message-Queuing Systems
2. Group Communication Systems

Group Communication Systems

Group Communication systems enable one-to-many communication

Multicast can be supported using two approaches

1. Network-level multicasting
2. Application-level multicasting

1. Network-Level Multicast

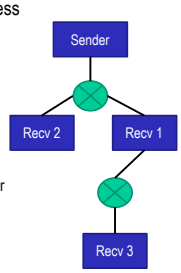
Each multicast group is assigned a unique IP address

Applications "join" the multicast group

Multicast tree is built by connecting routers and computers in the group

Network-level multicast is not scalable

- Each DS may have a number of multicast groups
- Each router on the network has to store information for multicast IP address for each group for each DS



2. Application-Level Multicast (ALM)

ALM organizes the computers involved in a DS into an overlay network

- The computers in the overlay network *cooperate* to deliver messages to other computers in the network

Network routers do not directly participate in the group communication

- The overhead of maintaining information at all the Internet routers is eliminated
- Connections between computers in an overlay network may cross several physical links. Hence, ALM may not be optimal

Summary

Several powerful and flexible paradigms to communicate between entities in a DS

- Inter-Process Communication (IPC)
  - IPC provides a low-level communication API
  - e.g., Socket API
- Remote Invocation
  - Programmer can transparently invoke a remote function by using a local procedure-call syntax
  - e.g., RPC and RMI
- Indirect Communication
  - Allows one-to-many communication paradigm
  - Enables space and time decoupling
  - e.g., Multicasting and Message-Queue systems

Next class

Naming in Distributed Systems

- Identify why entities have to be named
- Examine the naming conventions
- Describe name-resolution mechanisms