Communication in Distributed Systems – Fundamental Concepts

Hong-Linh Truong
Distributed Systems Group, Vienna University of Technology

truong@dsg.tuwien.ac.at
dsg.tuwien.ac.at/staff/truong
What is this lecture about?

- Understanding basic terminologies in communication in distributed systems
- Understanding key concepts in communication in distributed systems
Learning Materials

- Main reading:
  - Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall
    - Chapters 3 & 4
    - Chapters 2, 3, 7.
  - Craig Hunt, TCP/IP Network Administration, 3ed, 2002, O’Reilly.

- Test the examples in the lecture
  - Some code http://www.infosys.tuwien.ac.at/teaching/courses/VerteilteSysteme/exs/
Outline

- Communication entities, paradigm, roles/responsibilities
- Key issues in communication in distributed systems
- Protocols
- Processing requests
- Summary
COMMUNICATION ENTITIES, PARADIGM, AND ROLES
Hardware, software layer, programs

Software layer

Hardware (CPU, Memory, Network)

Operating Systems

Middleware/Libraries/Runtime systems

Applications

Programs/Programming Languages

- C/C++/Java, Python, ...
- Different types of programs: systems versus applications; sequential versus parallel ones; clients versus servers/services

Hardware heterogeneity

DS WS 2014
System Layers and Core OS functionality

Core OS functionality

Different OSs with a common middleware layer

Source: Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5

DS WS 2014
Within a non distributed OS

- Process – the program being executed by the OS
- Threads within a process
- Switching thread context is much cheaper than that for the process context
- Blocking calls in a thread do not block the whole process

Communication in distributed systems

- between processes within a single application/middleware/service
- among processes belonging to different applications/middleware/services
- Among computing nodes which have no concept of processes (e.g. sensors)

Q: Identify some concrete types of communication entities in real-world distributed systems (e.g., in a parallel cluster system)
Q: why is understanding time and space uncoupling important for implementing communication in distributed systems?
Communication Paradigm

- Interprocess communication
  - Low-level message-based communication, e.g., when communication entities are processes

- Remote invocation
  - (direct) calling of remote functions (of services/objects)

- Indirect communication
  - Communication carried out through third parties
Communication roles and responsibilities

- Several terms indicating communication entities
  - Objects, components, processes or services, clients, servers
  - forms versus roles/responsibilities

- Roles
  - Client/Server: client requests - server serves!
  - Sender/Receiver: w.r.t send/receive operation
  - Service: w.r.t. offering functionality
    - Network service, software-as-a-service,

Q: Can a service have multiple servers placed in different machines?
Communication networks in distributed systems

- Maybe designed for specific types of environments
  - High performance computing, M2M (Machine-to-Machine), building/home/city management, etc.
  - Events, voices, documents, image data, etc.

- Distributed, different network spans
  - Personal area networks (PANs), local area networks (LANs), campus area networks (CANs), metropolitan area networks (MANs), and wide area networks (WANs)
  - Communication entities are placed in different locations

- Different layered networks for distributed systems
  - Physical versus overlay network topologies (virtual network topologies atop physical networks)
Layered communication

In the view of P1 and P2

End-to-end process-to-process communication

E.g., email abc@tuwien.ac.at to ab@gmail.com

Sender/client

Communication Networks

Receiver/server

Holistic system view

End-to-end process-to-process communication

DS WS 2014
Q: What are the benefits of group communication? Give some concrete examples (e.g., in P2P and social networks).
Identifiers of entities participating in communication

- Communication cannot be done without knowing identifiers (names) of participating entities
  - Local versus global identifier
  - Individual versus group identifier
- Multiple layers/entities $\rightarrow$ different forms of identifiers
  - Process ID in an OS
  - Machine ID: name/IP address
  - Access point: (machine ID, port number)
  - A unique communication ID in a communication network
  - Emails for humans
  - Group ID
Examples of communication patterns (1)

- A User Agent wants to find a Service Agent
- Different roles and different communication patterns
- Get [http://jslp.sourceforge.net/](http://jslp.sourceforge.net/) and play samples to see how it works

DS WS 2014
Examples of communication patterns (2)

- MPI (Message Passing Interface)

```c
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
source=0;
count=4;
if(myid == source){
  for(i=0;i<count;i++)
    buffer[i]=i;
}
MPI_Bcast(buffer,count,MPI_INT,source,MPI_COMM_WORLD);
```

`$sudo apt-get install mpich$
`$mpicc c_ex04.c$
`$mpirun -np 4 ./a.out`

http://geco.mines.edu/workshop/class2/examples/mpi/c_ex04.c
Connection-oriented or connectionless communication

The message: „there is a party tonight“

P1 ————> P2

Write the message in a letter
Find the phone number of P2
Go to the post office
Call P2
Send the letter to P2
Tell P2 the message

Connection-oriented communication between P1 and P2 requires the setup of communication connection between them first – no setup in connectionless communication.

Q: What are the pros/cons of connection-oriented/connectionless communications? Is it possible to have a connectionless communication between (P1,P2) through some connection-oriented connections?
Blocking versus non-blocking communication calls

- Blocking: the process/thread execution is suspended until the message transmission finishes
- Non-blocking: the process/thread execution continues without waiting until the finish of the message transmission

Send: transmitting a message is finished, it does not necessarily mean that the message reaches its final destination.

Q: Analyze the benefits of non-blocking communication. How does non-blocking receive() work?
Persistent and transient communication

- **Persistent communication**
  - Messages are kept in the communication system until they are delivered to the receiver
  - Often storage is needed

- **Transient communication**
  - Messages are kept in the communication temporary only if both the sender and receiver are live
Asynchronous versus synchronous communication

- **Asynchronous:** the process continues after as soon as sending messages have been copied to the local buffer
  - Non blocking send; receive may/may not be blocking
  - Callback mechanisms

- **Synchronous:** the sender waits until it knows the messages have been delivered to the receiver
  - Blocking send/blocking receive
  - Typically utilize connection-oriented and keep-alive connection
  - Blocking request-reply styles
Q: How can we achieve the „persistent communication“? What are possible problems if a server sends an accepted/ACK message before processing the request?
Stateful versus Stateless Server

P1: sender/client

P2: receiver/server

T0

req1

P2 has no information about P1 before

T1

req2

Does P2 keep information about P1 from the previous request?

<table>
<thead>
<tr>
<th>Stateless server</th>
<th>Soft State</th>
<th>Stateful Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does not keep client‘s state information</td>
<td>Keep some limited client‘s state information in a limited time</td>
<td>Maintain client‘s state information permanently</td>
</tr>
</tbody>
</table>

Q: Give an example of a stateless communication built atop stateless communication. Analyze “web cookie” w.r.t. stateless/stateful support.
Handling out of band data

Normal case:
- P1 (sender/client) sends Req1, Req2 to P2 (receiver/server).
- All messages come to P2 in the same port, no clear information about priority.

Out of band data with a separate transmission port:
- P1 (sender/client) sends Req1, Req2 to P2 (receiver/server).
- Out-of-band data and communication channel for important data.

Q: How can out-of-band data and normal data be handled by using the same transmission channel?
COMMUNICATION PROTOCOLS
Some key questions – Protocols

Communication patterns
- Can I use a single sending command to send the message to multiple people?

Identifier/Naming/Destination
- How do I identify the guys I need to send the message

Connection setup
- Can I send the message without setting up the connection

Message structure
- Can I use German or English to write the message

Layered communication
- Do I need other intermediators to relay the message?

The message: „there is a party“ tonight

A communication protocol will describe rules addressing these issues
Applications and Protocols

Application-specific protocols

Application-independent protocols

Layered Communication Protocols

- Complex and open communication requires multiple communication protocols
- Communication protocols are typically organized into different layers: layered protocols/protocol stacks
- Conceptually: each layer has a set of different protocols for certain communication functions
  - Different protocols are designed for different environments/criteria
- A protocol suite: usually a set of protocols used together in a layered model
OSI – Open Systems Interconnection Reference Model

OSI Layers

- **Application**
  - Support application-specific needs

- **Presentation**
  - Process information format and deliver the information for the application layer (e.g., serializing and encryption)

- **Session**
  - Manage communication sessions between applications

- **Transport**
  - Provide an end-to-end communication for applications by delivering data among applications

- **Network**
  - Route data packets among senders/receivers

- **Data Link**
  - Deal with sending data frames (units of bits) and detecting/correcting data frames

- **Physical Layer**
  - Transfer binary data (bits) over physical interfaces (e.g., fiber optics)
How layered protocols work – message exchange

- Principles of constructing messages/data encapsulation

Examples of Layered Protocols

Application Layer
- FTP
- HTTP
- ModBus
- CoAP
- BACNet
- KNX (Konnex)

Presentation Layer
- TCP
- UDP
- RUDP
- SCTP
- IP
- Ethernet
- CSMA/CD
- CSMA/CA

Session Layer
- LonWorks

Transport Layer
- P-persistent CSMA

Network Layer
- X.21
- PL-20
- TP

Data Link Layer
- IEEE 802.3 (Ethernet)
- IEEE 802.11 (Wifi)
- ATM

Physical Layer
- ZigBee
TCP/IP

- The most popular protocol suite used in the Internet
- Four layers

**Protocol suite**

- Application Layer
  - Most network hardware
- Transport Layer
  - SMTP, HTTP, Telnet, FTP, etc.
- Internet Layer
- Link Layer
  - UDP, TCP
  - Internet Protocol (IP)
Internet Protocol (IP)

- Defines the datagram as the basic data unit
- Defines the Internet address scheme
- Transmits data between the Network Access Layer and Transport Layer
- Routes datagrams to destinations
- Divides and assembles datagrams

Figure source: http://en.wikipedia.org/wiki/Internet_protocol_suite
TCP/IP – Transport Layer

- Host-to-host transport features
- Two main protocols: TCP (Transmission Control Protocol) and UDP (User Datagram Protocol)

<table>
<thead>
<tr>
<th>Layer\Protocol</th>
<th>TCP</th>
<th>UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application layer</td>
<td>Data sent via Streams</td>
<td>Data sent in Messages</td>
</tr>
<tr>
<td>Transport Layer</td>
<td>Segment</td>
<td>Packet</td>
</tr>
<tr>
<td>Internet Layer</td>
<td>Datagram</td>
<td>Datagram</td>
</tr>
<tr>
<td>Link Layer</td>
<td>Frame</td>
<td>Frame</td>
</tr>
</tbody>
</table>

Note: pay attention with the terms „packet/datagram“ in TCP/IP versus that in the OSI model.
TCP operations

`$sudo nast -d -T iptest >ip.out`

`$wget www.tuwien.ac.at`

---(TCP)----------------------------------------------------------
192.168.1.7:46023(unknown) -> 128.130.35.76:80(http)
TTL: 64 Window: 14600 Version: 4 Length: 60
FLAGS: -S- - A- Seq: 3308581872 - ACK: 0
Packet Number: 16

---(TCP)----------------------------------------------------------
128.130.35.76:80(http) -> 192.168.1.7:46023(unknown)
TTL: 54 Window: 14480 Version: 4 Length: 60
FLAGS: -S--A- Seq: 3467332359 - ACK: 3308581873
Packet Number: 17

---(TCP)----------------------------------------------------------
192.168.1.7:46023(unknown) -> 128.130.35.76:80(http)
TTL: 64 Window: 115 Version: 4 Length: 52
FLAGS: -A- Seq: 3308581873 - ACK: 3467332360
Packet Number: 18

---(TCP)----------------------------------------------------------
192.168.1.7:46023(unknown) -> 128.130.35.76:80(http)
TTL: 64 Window: 115 Version: 4 Length: 166
FLAGS: -PA- Seq: 3308581873 - ACK: 3467332360
Packet Number: 19

---(TCP Data)----------------------------------------------------
GET / HTTP/1.1

---(TCP)----------------------------------------------------------
128.130.35.76:80(http) -> 192.168.1.7:46023(unknown)
TTL: 54 Window: 114 Version: 4 Length: 52
FLAGS: -A- Seq: 3467332360 - ACK: 3308581987
Packet Number: 20

---(TCP)----------------------------------------------------------
128.130.35.76:80(http) -> 192.168.1.7:46023(unknown)
TTL: 54 Window: 114 Version: 4 Length: 1500
FLAGS: -A- Seq: 3467332360 - ACK: 3308581987
Packet Number: 21

---(TCP Data)----------------------------------------------------
HTTP/1.1 200 OK

Communication protocols are not enough

- We need more than just communication protocols
  - E.g., resolving names, electing a communication coordinator, locking resources, and synchronizing time

- Middleware
  - Including a set of general-purpose but application-specific protocols, middleware communication protocols, and other specific services.
Middleware Protocols

HANDLING COMMUNICATION MESSAGES/REQUESTS
Where communication tasks take place?

- **Message passing – send/receive**
  - Processes *send* and *receive* messages
    - Sending process versus receiving process
    - Communication is done by using a set of functions for communication implementing protocols

- **Remote method/procedure calls**
  - A process *calls/invokes a (remote) procedure* in another process
    - Local versus remote procedure call, but in the same manner

- **Remote object calls**
  - A process *calls/invokes a (remote) object* in another process
Basic send/receive communication

# Echo client program
import socket

HOST = 'daring.cwi.nl'    # The remote host
PORT = 50007              # The same port as used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
s.send('Hello, world')
data = s.recv(1024)
s.close()
print 'Received', repr(data)

# Echo server program
import socket

HOST = ''                 # Symbolic name meaning the local host
PORT = 50007              # Arbitrary non-privileged port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print 'Connected by', addr
while 1:
    data = conn.recv(1024)
    if not data: break
    conn.send(data)
conn.close()

Python source: http://docs.python.org/release/2.5.2/lib/socket-example.html
Remote procedure calls

```c
void hello_prog_1(char *host)
{
    CLIENT *clnt;
    char *result_1;
    char *hello_1_arg;

    #ifndef DEBUG
    clnt = clnt_create(host, HELLO_PROG, HELLO_VERS, "udp");
    if (clnt == NULL)
    {
        clnt_pcreateerror(host);
        exit (1);
    }
    #endif /* DEBUG */

    result_1 = hello_1((void *)&hello_1_arg, clnt);
    if (result_1 == (char **) NULL)
    {
        clnt_perror(clnt, "call failed");
    }

    #ifndef DEBUG
    clnt_destroy(clnt);
    #endif /* DEBUG */

    printf("result is: %sn", (*result_1));
}

int main(int argc, char *argv[])
{
    char *host;

    if (argc < 2)
    {
        printf("usage: %s server_host\n", argv[0]);
        exit(1);
    }

    host = argv[1];
    hello_prog_1 (host);
    exit(0);
}
```

char ** hello_1_svc(void *argp, struct svc_req *rqstp)
{
    static char * result ="Hello"
;
    /* insert server code here */

    return &result;
}
Remote object calls

Java source: http://docs.oracle.com/javase/tutorial/rmi/overview.html
Processing multiple requests

- How to deal with multiple, concurrent messages received?

- Problems:
  - Different roles: clients versus servers/services
    - A large number of clients interact with a small number of servers/services
    - A single process might receive a lot of messages at the same time

- Impacts
  - performance, reliability, cost, etc.
Iterative versus concurrent processing

**Iterative processing**

1. Receive a request
2. Process the request
3. Return the result

**Concurrent processing**

1. Receive a request
2. Ask another process/thread to process the request
3. Wait for new requests

Request handling flow
Q: How does this model help to improve performance and fault-tolerance? What would be a possible mechanism to reduce costs based on the number of client requests?
Get a small test
- Download haproxy, e.g.
  $ sudo apt-get install haproxy
- Download SimpleHelloHTTPServer.java and haproxy configuration
- Run 1 haproxy instance and 3 http servers
  - Modify configuration and parameters if needed
- Run a test client
Using multiple threads

Q: Compare this architectural model with the super-server model?

Using message brokers/space repository

Example

- Get a free instance of RabbitMQ from cloudamqp.com
- Get code from: https://github.com/cloudamqp/java-amqp-example
- First run the test sender, then run the receiver

```java
channel.queueDeclare(QUEUE_NAME, false, false, false, null);
for (int i=0; i<100; i++) {
    String message = "Hello distributed systems guys: "+i;
    channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
    System.out.println("[x] Sent " + message + "");
    new Thread().sleep(5000);
}

while (true) {
    QueueingConsumer.Delivery delivery = consumer.nextDelivery();
    String message = new String(delivery.getBody());
    System.out.println("[x] Received "+ message + ");
}
```

Note: I modified the code a bit
Summary

- Complex and diverse communication patterns, protocols and processing models
- Choices are based on communication requirements and underlying networks
  - Understand their pros/cons
  - Understand pros and cons of their technological implementations
- Don’t forget to play with some simple examples to understand existing concepts
Thanks for your attention

Hong-Linh Truong
Distributed Systems Group
Vienna University of Technology
truong@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at/staff/truong