Communication in Distributed Systems – Programming

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?

- Examine and study main frameworks, libraries and techniques for programming communication in distributed systems
- Understand pros and cons of different techniques for different layers and purposes
- Be able to select the right solutions for the right systems
- Be able to combine different techniques for a complex problem
Learning Materials

- Main reading:
  - Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall
    - Chapters 3 & 4
  - George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair, "Distributed Systems – Concepts and Design“, 5nd Edition
    - Chapters 4,5,6 and 9
    - Chapter 15

- Papers referred in the lecture
  - Test the examples in the lecture
Outline

- Recall
- Message-oriented Transient Communication
- Message-oriented Persistent Communication
- Remote Invocation
- Web Services
- Streaming data programming
- Group communication
- Gossip-based Data Dissemination
- Summary
Recall

- One-to-one versus group communication
- Transient communication versus persistent communication
- Message transmission versus procedure call versus object method calls
- Physical versus overlay network
MESSAGE-ORIENTED TRANSIENT COMMUNICATION
Message-oriented Transient Communication at Transport Layer

- How does an application use the transport layer communication to send/receive messages?

Transport-level socket programming via socket interface

- Socket interface – Socket APIs
  - Very popular, supported in almost all programming languages and operating systems
  - Berkeley Sockets (BSD Sockets)
    - Java Socket, Windows Sockets API/WinSock, etc.
  - Designed for low-level system, high-performance, resource-constrained communication
What is a socket: a communication end point to/from which an application can send/receive data through the underlying network.

- **Client**
  - Connect, send and then receive data through sockets

- **Server:**
  - Bind, listen/accept, receive incoming data, process the data, and send the result back to the client

Q: Which types of information are used to describe the identifier of the “end point”?
### Socket Primitives

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>Create a new communication end point</td>
</tr>
<tr>
<td>Bind</td>
<td>Attach a local address to a socket</td>
</tr>
<tr>
<td>Listen</td>
<td>Announce willingness to accept connections</td>
</tr>
<tr>
<td>Accept</td>
<td>Block caller until a connection request arrives</td>
</tr>
<tr>
<td>Connect</td>
<td>Actively attempt to establish a connection</td>
</tr>
<tr>
<td>Send</td>
<td>Send some data over the connection</td>
</tr>
<tr>
<td>Receive</td>
<td>Receive some data over the connection</td>
</tr>
<tr>
<td>Close</td>
<td>Release the connection</td>
</tr>
</tbody>
</table>

Client-server interaction

Connection-oriented communication interaction

Server

| socket | bind | listen | accept | read | write | close |

Client

| socket | connect | write | read | close |

Synchronization point

Communication


Q: How can a multi-threaded server be implemented?
Example

- Simple echo service
  - Client sends a message to a server
  - Server returns the message
- Source code: https://github.com/tuwiendsg/distributedsystems/examples/tree/master/SimpleEchoSocket

Q: What if connect() happens before listen()/accept()?
Message-oriented Transient Communication at the Application level

Complex communication, large-scale number processes in the same application

Why are transport level socket programming primitives not good enough?
Message-passing Interface (MPI)

- Designed for parallel processing: http://www.mpi-forum.org/
- Well supported in clusters and high performance computing systems
- One-to-one/group and synchronous/asynchronous communication

Basic MPI concepts

- Communicators/groups to determine a set of processes that can be communicated: MPI_COMM_WORLD represents all mpi processes
- Rank: a unique identifier of a process
- A set of functions to manage the execution environment
- Point-to-point communication functions
- Collective communication functions
- Functions handling data types
## Message-passing Interface (MPI)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Init</td>
<td>Initialize the MPI execution environment</td>
</tr>
<tr>
<td>MPI_Comm_size</td>
<td>Determine the size of the group given a communicator</td>
</tr>
<tr>
<td>MPI_Comm_rank</td>
<td>Determine the rank of the calling process in group</td>
</tr>
<tr>
<td>MPI_Send()</td>
<td>Send a message, blocking mode</td>
</tr>
<tr>
<td>MPI_Recv()</td>
<td>Receive a message, blocking mode</td>
</tr>
<tr>
<td>…</td>
<td></td>
</tr>
<tr>
<td>MPI_Bcast()</td>
<td>Broadcast a message from a process to others</td>
</tr>
<tr>
<td>MPI_Reduce()</td>
<td>Reduce all values from all processes to a single value</td>
</tr>
<tr>
<td>…</td>
<td></td>
</tr>
<tr>
<td>MPI_Finalize()</td>
<td>Terminate the MPI execution environment</td>
</tr>
</tbody>
</table>
Example

```c
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
if (myid == 0) {
    printf("I am %d: We have %d processors\n", myid, numprocs);
    sprintf(output, "This is a message sending from %d", i);
    for (i=1;i<numprocs;i++)
        MPI_Send(output, 80, MPI_CHAR, i, 0, MPI_COMM_WORLD);
}
else {
    MPI_Recv(output, 80, MPI_CHAR, i, 0, MPI_COMM_WORLD, &status);
    printf("I am %d and I receive: %s\n", output);
}

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);
    for (i=0;i<count;i++) {
        printf("I am %d and I receive: %d \n", myid, buffer[i]);
    }
} else {
    MPI_Recv(output, 80, MPI_CHAR, i, 0, MPI_COMM_WORLD, &status);
    printf("I am %d and I receive: %s\n", myid, output);
}

printf("\n");
MPI_Finalize();
```

Code: https://github.com/tuwiendsg/distributedsystemsexamples/tree/master/mpi-ex
MESSAGE-ORIENTED PERSISTENT COMMUNICATION
Message-oriented Persistent Communication – Queuing Model

- Message-queuing systems or Message-Oriented Middleware (MOM)
- Well-supported in large-scale systems for
  - Persistent but asynchronous messages
  - Scalable message handling
  - Different communication patterns
- Several Implementations
Message-oriented Persistent Communication – Queuing Model

Communication models with time (un)coupling

(a) Sender running, Receiver running
(b) Sender running, Receiver passive
(c) Sender passive, Receiver running
(d) Sender passive, Receiver passive

Q1: Give an example of case (d)

Operations
PUT
GET
POLL
NOTIFY

Message-oriented Persistent Communication – Queuing Model


Message Brokers

- **Publish/Subscribe**: messages are matched to applications
- **Transform**: messages are transformed from one format to another one suitable for specific applications

Example – Advanced Message Queuing Protocol (AMQP)

- http://www.amqp.org

Apache Qpid™

RabbitMQ

stormmq
Content-Based Message Routing:

AMQP

Note: defined in AMQP 0-10
But not in AMQP 1.0

Example: AMQP

```java
ConnectionFactory factory = new ConnectionFactory();
    factory.setUri(uri);
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

    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);
    }

    channel.close();
    connection.close();
```

Source code:
https://github.com/cloudamqp/java-amqp-example, see also the demo in the lecture 2
REMOTE INVOCATION
Remote Procedure Call

How can we call a procedure in a remote process in a similar way to a local procedure?

Remote Procedure Call (RPC): hides all complexity in calling remote procedures

- Well support in many systems and programming languages

Q1: Which types of applications are suitable for RPC?

Message format and data structure description

- Passing parameters and results needs agreed message format between a client and a server.

Marshaling/unmarshalling describes the process of packing/unpacking parameters into/from messages (note: encoding/decoding are also the terms used).

- Data types may have different representations due to different machine types (e.g., SPARC versus Intel x86).

Interface languages can be used to describe the common interfaces between clients and server.
Generating stubs

- Interface description (e.g., IDL, XML)
- Message Format (e.g., XDR, XML)
- Transport information (e.g., HTTP, TCP, UDP)
- Stubs: Code for marshalling/unmarshalling
Detailed Interactions

1. Client call to procedure

2. Stub builds message

3. Message is sent across the network

4. Server OS hands message to server stub

5. Stub unpacks message

6. Stub makes local call to "add"

One-way RPC

Client

Server

Message

Call local procedure

Call and continue

Time
Recall: (A)synchronous communication
Q1: How can asynchronous RPC be implemented

Asynchronous RPC

Two asynchronous RPCs/ Deferred synchronous RPC

Q: List some possible failures in RPC interactions.

Some RPC implementations

- rpcgen – SUN RPC
  - IDL for interface description
  - XDR for messages
  - TCP/UDP for transport

- XML-RPC
  - XML for messages
  - HTTP for transport

- JSON-RPC
  - JSON for messages
  - HTTP and/or TCP/IP for transport

Remote Method Invocation/Remote Object Call

- Remote object method invocation/call
  - RPC style in object-oriented programming

Diagram:
- RMI Client
- locate objects
- RMI Registry
- publish objects
- RMI Server
- invoke object methods
- obj
- obj
- obj
- obj
Example of RPC

```
program ADD_PROG {
    version ADD_VERS {
        int add(int i, int j) = 1;
    } = 1;
    } = 0x23452345;
```

Code: https://github.com/tuwiendsg/distributedsystemsexamples/tree/master/rpcadd-ex
WEB SERVICES
Web services (1)

- Service: common software functionalities/capabilities offered through well-defined interfaces and consistent usage policies
- Socket APIs, RPC, or RMI can be used to implement „services“, but
  - Do not work very well in the Web/Internet environment
  - Do not support well the integration of different software systems

Web Services: “A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.” -- http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#whatis
Web services (2)

<table>
<thead>
<tr>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Services</td>
</tr>
<tr>
<td>SOAP/WSDL</td>
</tr>
<tr>
<td>URIs</td>
</tr>
</tbody>
</table>

Services and descriptions
Protocols/interfaces
Identifiers, data format, transportation

- Why Web services are important in distributed systems?
  - Support interoperability
  - Hide system complexity and implementation detail
  - Enable easy integration of diverse and distributed software components

XML-based Web service communication protocols

- Through runtime, clients and services can send and receive SOAP messages → different communication patterns
  - SOAP messages (XML-based) like an envelope with a header and a body
- SOAP messages are transported using different transport protocols
- WSDL is used to describe a Web service
- Usually a Web service is hosted in an application server/container, which supports complex messages dispatching and handling
Architectural Design - REST

- Resources are identified and accessed through URIs
- Resources are data and functionality
- A Web service manages a set of resources
- A client and a service exchange representations of resources via standardized interface and protocols
  - Assume one-to-one communication/client-server model

Diagram:

- Web Service Client
  - GET (list/retrieve)
  - PUT (update/create)
  - POST (create/update)
  - DELETE (remove)
- Web Service
  - URI:\ Resource_i
  - URI_k:\ Resource_k
Web Services programming

- From WSDL to code, e.g.,
  - Java API for XML Web Services (JAX-WS)
    - Generate Web service stubs from WSDL files
      - E.g., wsdl2java

- Using annotations
  - XML-based Web services (SOAP)
    - JAX-WS annotations (JSR 181, JSR 224)
    - @WebService, @WebMethod

- REST
  - Java API for RESTful Web Services, JSR-311
    - @Path, @GET, @POST, …

- Well-supported in many programming languages
**JAX-WS**

```java
import javax.ws.WebService;

@WebService(endpointInterface = "demo.hw.server.HelloWorld",
            serviceName = "HelloWorld")
public class HelloWorldImpl implements HelloWorld {
    Map<Integer, User> users = new LinkedHashMap<Integer, User>();

    public String sayHi(String text) {
        System.out.println("sayHi called");
        return "Hello " + text;
    }

    public String sayHiToUser(User user) {
        System.out.println("sayHiToUser called");
        users.put(users.size() + 1, user);
        return "Hello " + user.getName();
    }

    @GET
    @Path("/test")
    @Produces(MediaType.TEXT_PLAIN)
    public String test() {
        return "Test working";
    }

    @PUT
    @Path("/{id}/unDemandControl/unhealthy")
    @Consumes("plain/text")
    public void checkUnHealthyState(String servicePartID, @PathParam("id") String id) {
        controlCoordination.triggerHealthFixServicePart(servicePartID, servicePartID);
    }

    @PUT
    @Path("/processAnnotation")
    @Consumes("application/xml")
    public void processAnnotation(String serviceID, String entity, SYBLAnnotation annotation) {
        controlCoordination.processAnnotation(serviceID, entity, annotation);
    }

    @PUT
    @Path("/descriptionInternalModel")
    @Consumes("application/xml")
    public void setApplicationDescriptionInfoInternalModel(String applicationDescriptionXML, String applicationDescription)
        controlCoordination.setApplicationDescriptionInfoInternalModel(applicationDescription);
```


**REST**

```java
GET
@Path("/test")
@Produces(MediaType.TEXT_PLAIN)
public String test() {
    return "Test working";
}

@PUT
@Path("/{id}/unDemandControl/unhealthy")
@Consumes("plain/text")
public void checkUnHealthyState(String servicePartID, @PathParam("id") String id) {
    controlCoordination.triggerHealthFixServicePart(servicePartID, servicePartID);
}

@PUT
@Path("/processAnnotation")
@Consumes("application/xml")
public void processAnnotation(String serviceID, String entity, SYBLAnnotation annotation) {
    controlCoordination.processAnnotation(serviceID, entity, annotation);
}

@PUT
@Path("/descriptionInternalModel")
@Consumes("application/xml")
public void setApplicationDescriptionInfoInternalModel(String applicationDescriptionXML, String applicationDescription)
    controlCoordination.setApplicationDescriptionInfoInternalModel(applicationDescription);
```

Applications – Cloud Computing

- Cloud resources
  - Files, storage, compute machines, middleware, etc.
  - Resources offered via RESTful models
- Many cloud services support REST APIs
- Examples

<table>
<thead>
<tr>
<th>Compute &amp; Networking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon EC2</td>
</tr>
<tr>
<td>Auto Scaling</td>
</tr>
<tr>
<td>Elastic Load Balancing</td>
</tr>
<tr>
<td>Amazon VPC</td>
</tr>
<tr>
<td>Amazon Route 53</td>
</tr>
<tr>
<td>AWS Direct Connect</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Storage &amp; Content Delivery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon S3</td>
</tr>
<tr>
<td>Amazon Glacier</td>
</tr>
<tr>
<td>Amazon EBS</td>
</tr>
<tr>
<td>AWS Import/Export</td>
</tr>
<tr>
<td>AWS Storage Gateway</td>
</tr>
<tr>
<td>Amazon CloudFront</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon RDS</td>
</tr>
<tr>
<td>Amazon DynamoDB</td>
</tr>
<tr>
<td>Amazon ElastiCache</td>
</tr>
<tr>
<td>Amazon Redshift</td>
</tr>
<tr>
<td>Amazon SimpleDB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon EMR</td>
</tr>
<tr>
<td>Amazon Kinesis</td>
</tr>
<tr>
<td>AWS Data Pipeline</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deployment &amp; Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Identity &amp; Access Management</td>
</tr>
<tr>
<td>AWS CloudTrail</td>
</tr>
<tr>
<td>Amazon CloudWatch</td>
</tr>
<tr>
<td>AWS Elastic Beanstalk</td>
</tr>
<tr>
<td>AWS CloudFormation</td>
</tr>
<tr>
<td>AWS OpsWorks</td>
</tr>
<tr>
<td>AWS CloudHSM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>App Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon AppStream</td>
</tr>
<tr>
<td>Amazon CloudSearch</td>
</tr>
<tr>
<td>Amazon Elastic Transcoder</td>
</tr>
<tr>
<td>Amazon SES</td>
</tr>
<tr>
<td>Amazon SOS</td>
</tr>
<tr>
<td>Amazon SWF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mobile Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Cognito</td>
</tr>
<tr>
<td>Amazon Mobile Analytics</td>
</tr>
<tr>
<td>Amazon SNS</td>
</tr>
<tr>
<td>AWS Mobile SDK for Android</td>
</tr>
<tr>
<td>AWS Mobile SDK for iOS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Getting Started with AWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>macOS</td>
</tr>
<tr>
<td>Computing Basics (Linux)</td>
</tr>
<tr>
<td>Computing Basics (Windows)</td>
</tr>
<tr>
<td>Web App Hosting (Linux)</td>
</tr>
<tr>
<td>Web App Hosting (Windows)</td>
</tr>
<tr>
<td>Deploying a Web Application</td>
</tr>
<tr>
<td>Analyzing Big Data with AWS</td>
</tr>
<tr>
<td>Static Website Hosting</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tools for Amazon Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Management Console</td>
</tr>
<tr>
<td>AWS SDK for Java</td>
</tr>
<tr>
<td>AWS SDK for JavaScript</td>
</tr>
<tr>
<td>AWS SDK for .NET</td>
</tr>
<tr>
<td>AWS SDK for PHP</td>
</tr>
<tr>
<td>AWS SDK for Python (boto)</td>
</tr>
<tr>
<td>AWS SDK for Ruby</td>
</tr>
<tr>
<td>AWS Toolkit for Eclipse</td>
</tr>
<tr>
<td>AWS Toolkit for Visual Studio</td>
</tr>
<tr>
<td>AWS Command Line Interface</td>
</tr>
<tr>
<td>AWS Tools for Windows PowerShell</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Additional Software &amp; Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexa Top Sites</td>
</tr>
<tr>
<td>Alexa Web Information Service</td>
</tr>
<tr>
<td>Amazon Mechanical Turk</td>
</tr>
<tr>
<td>Amazon Silk</td>
</tr>
</tbody>
</table>
STREAMING DATA PROGRAMMING
Data stream programming

Data stream: a sequence of data units

e.g. reading bytes from a file and send bytes via a TCP socket

- Data streams can be used for
  - Continuous media (e.g., video)
  - Discrete media (e.g., stock market events/twitter events)
Timing issues

- **Asynchronous** transmission mode
  - no constraints on when the transmission completes

- **Synchronous** transmission mode:
  - maximum end-to-end delay defined for each data unit

- **Isochronous** transmission
  - maximum and minimum end-to-end delay defined

When the transmission of m2 completes
Multiple streams

Complex stream/multiple streams data processing

Tools

Esper  Storm  S4  Gigaspaces XAP  Streambase
Example: Complex event processing with Esper

http://esper.codehaus.org/esper.

Streaming event data

public class InteractionEvent {
    public final static String REQUEST = "Request";
    public final static String RESPONSE = "Response";
    private String clientEndpoint=null;
    private String activityURI=null;
    private String serviceEndpoint=null;
    private String messageCorrelationID=null;
    private String messageType=null;
    //...
}

EPL (Event Processing Language)

select clientEndpoint, serviceEndpoint from InteractionEvent.win:length(100) where messageType="Request"

Esper Runtime Engine

ResultHandler

public class NumberCallHandler extends BaseResultHandler {
    @Override
    public void update(Map[] insertStream, Map[] removeStream) {
        //....
    }
}
Group communication

- Group communication use multicast messages
  - E.g., IP multicast or application-level multicast

**Atomic Multicast:** Messages are received either by every member or by none of them

**Reliable multicast:** messages are delivered to all members in the best effort – but not guaranteed.
Atomic Multicast

Q1: Give an example of atomic multicast

Example of implementing multicast using one-to-one communication

Sender’s program

\[
\begin{align*}
&i := 0; \\
&\textbf{do} \ i \neq n \rightarrow \\
&\quad \text{send message to member}[i]; \\
&\quad i := i + 1 \\
&\textbf{od}
\end{align*}
\]

Receiver’s program

\[
\begin{align*}
&\textbf{if} \ m \text{ is new} \rightarrow \\
&\quad \text{accept it; } \\
&\quad \text{multicast } m; \\
&\quad \textbf{if} \ m \text{ is duplicate} \rightarrow \text{discard } m
\end{align*}
\]


Q2: How do we know “m is new”?
Application processes are organized into an overlay network, typically in a mesh or a tree.
Application-level Multicast Communication (2)

Gossip-based Data Dissemination

Why gossip? E.g., https://www.youtube.com/watch?v=OPYhk_NbEtA#t=22

It can spread messages fast and reliably

Active thread (peer P):

1. selectPeer(&Q);
2. selectToSend(&bufs);
3. sendData(Q, bufs);
4. 
5. receiveFrom(Q, &bufr);
6. selectToKeep(cache, bufr);
7. processData(cache);

Passive thread (peer Q):

1. 
2. 
3. receiveFromAny(&P, &bufr);
4. selectToSend(&bufs);
5. sendData(P, bufs);
6. selectToKeep(cache, bufr);
7. processData(cache)

Gossip-based Data Dissemination (2)

- Give a system of \(N\) nodes and there is the need to send some data items
- Every node has been updated for data item \(x\)
  - Keep \(x\) in a buffer whose maximum capability is \(b\)
  - Determine a number of times \(t\) that the data item \(x\) should be forwarded
  - Randomly contact \(f\) other nodes (the fant-out) and forward \(x\) to these nodes

Different configurations of \((b,t,f)\) create different algorithms

Summary

- Various techniques for programming communication in distributed systems
  - Transport versus application level programming
  - Transient versus persistent communication
  - Procedure call versus messages
  - Web Services
  - Streaming data
  - Multicast and gossip-based data dissemination
- 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