Today: Distributed Systems

What gets harder when we move from a stand-alone system to a distributed environment?

Unless otherwise stated, assume a reliable end-to-end message delivery.

- Failure recovery
- Deadlock detection and recovery
- Critical sections
- Timing (e.g., synchronization)
- Resource sharing

| Insecure | Insecure
| Syncрушные | Syncрушные
| Process to process (routing) | Machine to machine
| Reliable | Unreliable
| Ordered | Unordered (sometimes)
| Arbitrary size | Limited size

Abstract: Messages

Physical reality: Packets

Processes in a distributed system all communicate via a message exchange.
we need a quantitative analysis to decide where the cutoffs are.

Reality is usually in the middle somewhere.

Locally:
  • If communication is slow and expensive, we should do most processing
  • distributed environment:
    • If communication is fast and cheap, we can utilize all the resources in the
  • Computation versus Communication

| Computation Sharing |

| Incompatible. |

The fundamental tradeoff in resource sharing is to complete user

| Job Migration: moving the job (computation and data) or part of the job |
| Computation Migration: move the computation to the data |
| Data Migration: moving the data around |

There are many mechanisms for sharing (hardware, software, data) resources.
A sends a message to the predefined process at B, which performs the requested
\( \text{Remote Procedure Call (RPC)} \): Suppose A wants to access file at site B.

**Example:** A small program which produces a short summary of a large file, or a database query

\( \text{Computation Migration} \) may occur when it is more efficient to transfer the

<table>
<thead>
<tr>
<th>Computation Migration</th>
</tr>
</thead>
</table>

\( \text{2. Keep file at B, access file remotely from A} \)

- Single file server can be a performance bottleneck
- Single copy of file, so no consistency problems
- Converting the file from A's format to B's may be difficult to do in pieces
- Saves the transfer cost

\( \text{1. Copy file to process A} \)

- All subsequent accesses at A are local
- Multiple copies can cause consistency problems
- Data must be converted to A's data format
- Costly if the file is large

\( \text{Data Migration} \) may occur when process at site A accesses a file at site B.

<table>
<thead>
<tr>
<th>Data Migration</th>
</tr>
</thead>
</table>
- RPC is one common way this structure is implemented.
  - The service sends the server a request to perform some action. The server sends
    a response.
- A client first binds to the server, e.g., locates it in the network and establishes a
  connection.
- A server may exist on one or more nodes.
- A client is a program that uses the service.
- The service, the database service, etc.
- A service is a process or collection of processes that provide a service, e.g., name
  resolution.

One of the most common models for structuring distributed computation

Client/Server Model

Moving the data and computation.

- Job Migration: perform the job (or parts of the job) at remote sites by
  load balancing.
- Hardware preference: Jobs may match a specific piece of hardware somewhere in the
  system.
- Compartmental special: concurrent (parallel) execution of parts of the job.
- Software preference: Jobs may require software only available on a specific site. For
  example, the specific license of expensive software.
- User interaction: may want to hide migration from the user. For example, in load
  balancing, may want the user to specify migration (hardware/software preferences)
Remote Procedure Call: Implementation Issues

- The RPC mechanism uses the procedure signature (number and type of arguments and return value) for each procedure on which we want to support RPC.

Basic Idea:

- OS manages the communication.
- To use the server, the client does a procedure call.
- Servers export procedures for some set of clients to call.
Remote Procedure Call

In most RPC systems, dynamic binding is performed using a name service.

- The client, before issuing any calls, asks the name service for the location of a server.
- When the server starts up, it exports its interface and identity (e.g., to a network).

How does the client know the right port?

Comparison between RPC and a regular procedure call:

**Server Side:**

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Create</td>
<td>Create threads</td>
</tr>
<tr>
<td>2. Wait</td>
<td>Wait for a command</td>
</tr>
<tr>
<td>3. Unpack</td>
<td>Unpack request parameters</td>
</tr>
<tr>
<td>4. Call</td>
<td>Call procedure with thread</td>
</tr>
<tr>
<td>5. Build</td>
<td>Build message</td>
</tr>
<tr>
<td>6. Send</td>
<td>Send message</td>
</tr>
</tbody>
</table>

**Return Result**

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Send reply</td>
<td>Send reply with result(s)</td>
</tr>
<tr>
<td>2. Wait</td>
<td>Wait for response</td>
</tr>
<tr>
<td>3. Unpack</td>
<td>Unpack request parameters</td>
</tr>
<tr>
<td>4. Call</td>
<td>Call procedure with thread</td>
</tr>
</tbody>
</table>

**Remote Procedure Call: Implementation Issues**
- Should handle RemoteException
- Uses normal method calls for remote methods
- Looks up the server in the remote object registry

**Client**

- Registers the objects with the remote object registry
  - constructed as a subclass of RemoteObject

**Server**

- Creates one or more server objects - normal constructor call with the object being
  - Main program for server
    - implements each of the methods in the interface
  - Defines an interface listing the signatures of methods the server will satisfy

---

**Example: Server in Java**

- Interprets packets
  - **Remote Interface:** Generates client and server stubs that create and
  - **Implementation Class:** Server-side name server

**Example: Remote Method Invocation (RMI) in Java**

- Named: Java provides the following tools:

  - **Remote Interface:** Exports the interface automatically when the server
    - **Interface:** Supports references to non-replicated remote

**Example:**

- **Remote Interface:** Remote Interface (RemoteName)
  - Remote object that
    - contains a RemoteObject that binds a server to
  - Provides the following interface:

---
Example: Hello World Server

```
package example.hello;

public class HelloServer implements HelloWorld {

    public String sayHello() throws RemoteException {
        return "Hello World!";
    }

    public static void main(String[] args) {
        try {
            ServerSocketFactory serverSocketFactory = ServerSocketFactory.getDefault();
            ServerSocket serverSocket = serverSocketFactory.createServerSocket(1234);
            Socket socket = serverSocket.accept();
            PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
            writer.println("Hello World!");
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Example: Hello World Interface

```
package example.hello;

public interface HelloWorld {
    String sayHello() throws RemoteException;
    void sayBye() throws RemoteException;
}
```

Declare the methods that the server provides.
Example: Hello World Client

```
package example.bello;

import java.applet.Applet;
import java.awt.*;

public class HelloServer extends java.applet.Applet {

    public void init() {
        System.setSecurityManager(new SecurityManager());
    }

    public void doGet(String... arg) {
        System.out.println("Hello, World Server (contld)");
    }

    public void doPost(String... arg) {
        System.out.println("Hello, World Server (contld)");
    }
}
```

Example: Hello World Client (contld)
Applications running in different address spaces
- RPC is commonly used even on a single node for communication between services
- Relies on a stub compiler to automatically produce client/server stubs from the
  - RPC is essential language support for distributed programming.
- Most common model for communicating in distributed applications.

Mechanism: RPC

Client-Server Model

Data, computation, job migration

Summary

Example: Hello World Client (contd)