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

Processes vs. Machines

Machine to Machine

Reliable

Unreliable

Ordered (sometimes)

Unordered

Limited Size

Physical reality: Packets

Abstraction: Messages

Processes in a distributed system all communicate via a message exchange.

Failure recovery

Deadlock detection and recovery

Critical sections

Timing (e.g. synchronization)

Resource sharing

Last Class: Network Overview
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**

+ Incompatible.
  + Instructions as fast and as cheaply as possible (Fast and cheap are usually
    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.

**Resource Sharing**
action, and sends the result back to A. A sends a message to the prepared process at B, which performs the requested

Remote Procedure Calls (RPC): Suppose A wants to access file at site B.

say "wc" or a database query

Example: a small program which produces a short summary of a large file,

Computation Migration may occur when it is more efficient to transfer the

Computation Migration

2. Keep file at B, access file remotely from 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

I. Copy file to process A

Data Migration may occur when process at site A accesses a file at site B

Data Migration
- \( M \) is one common way this structure is implemented.  
  - back a response
  - connection: The client then sends the server a request to perform some action. The server sends a response to the client.  
  - A client first binds to the server, i.e., locates it in the network and establishes a connection.  
  - A server may exist on one or more nodes.  
  - The server may run one or more services, e.g., service, database services, etc.

One of the most common models for structuring distributed computation is by using the client/server paradigm:

**Client/Server Model**

Moving the data and computation:* Job Migration: perform the job (or parts of the job) at remote sites by load balancing. They want the user to specify migration (hardware/software preferences)
Remote Procedure Call

Remote Procedure Call: Implementation Issues

- Each procedure on which we want to support RPs
- The RPs mechanism uses the procedure signature (number and type of arguments and return value)
- To get a client stub that bundles up the RPC arguments and
- Packets are sent to or from the server, and

Diagram:

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 identities itself to a network.

#### How does the client know the right port?

#### Comparison between RPC and a regular procedure call

---

**Server Side:**

end loop

send reply

build reply with result

call procedure with parameters

wait for a command

create threads

**Client Side:**

wait for response

send message

build message

return result
- Should handle RemoteException
- Uses normal method call syntax for remote methods
- Looks up the server in the remote object registry

**Client**

- Registers the objects with the remote object registry
  * Constructor is a subclass of RemoteObject
- Creates one or more server objects - normal constructor call where 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

**Server**

---

**Example: Server in Java**

import java.rmi;

public class Server implements ServerInterface {

private ServerInterface server;

public Server() {
    // Initialize server
}

public void method1() {
    // Implement method
}
}

Server server = new Server();
server.bindRemoteObject("Server", server);

---

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

import java.rmi.

Remote server = Naming.lookup("Server");
server.method1();
Example: Hello World Server

```java
package example; // Declare the methods that the server provides:

public interface Hello World Interface
{

    public interface Hello extends java.rmi.Remote; // All servers must extend the Remote Interface.

    // The super constructor exports the interface and sets a port
    public HelloWorld() throws RemoteException
    { super(); }

    public String sayHello() throws RemoteException
    { return "Hello World!"; } // This is the "serialVersionUID" property.

    // The superclass constructor exports the interface and sets a port
    public HelloWorld() throws RemoteException
    { super(); }
}

import java.rmi.server.UnicastRemoteObject;
import java.rmi.*;
package example.hello; // Import Java.rmi.*;
```

Example: Hello World Server
module.exports = function hello() {
  console.log('Hello world!');
};

module.exports = function hello() {
  console.log('Hello world!');
};

import java.lang.*;

package example.hello;

{ // Example: Hello World
  String message = "Hi"
}

import java.lang.*;

package example.hello;

{ // Example: Hello World (contd)
  String message = "Hi"
}
Applications running in different address spaces.
- RPC is commonly used even on a single node for communication between
  files.
- Relies on a stub compiler to automatically produce client/server stubs from the
  stub.
- RPC is essential for language support for distributed programming.
- Most common model for communications in distributed applications.

Mechanism: RPC

Client-Server Model

Data, computation, job migration

Summary

Example: Hello World Client (contd)