CMPSCI 377 Operating Systems **Spring 2010**

Lecture 24: April 22

Lecturer: Prashant Shenoy *TA: Vimal Mathew & Tim Wood*

24.1 Challenges in Distributed Systems

The transition from running software within a single system to using a distributed environment adds a number of challenges.

Sharing resources among nodes in a distributed system is a challenge since different nodes may have different requirements, and they may be separated by long distances. Nodes may need to share data, and it is not always clear whether it is best to transfer the data or to transfer the software which must operate on the data. **Data Migration** refers to systems that transfer data between nodes in order to let different components operate on the data. In **Computation Migration**, the data is kept in one location, but the software operating on it is moved around. This can be useful when the amount of data that would need to be transferred is very large. Finally, **Job Migration** involves moving both the computation and the data between nodes. This can be used for a variety of reasons such as balancing load throughout the system or providing computational speedup by running the process on multiple nodes.

Determining whether to move data or computation often depends on the relative cost of **computation versus communication**. If communication is very cheap (e.g. within a LAN), then it can be beneficial to do frequent communication. On the other hand, in a WAN environment where communication is slow, it is often best to do most processing locally and only periodically transmit messages between hosts.

24.2 Client/Server Model

One of the most common models for building a distributed system is the Client/Server model. In these systems, there is generally a single node that acts as a **server**, running software which provides some sort of service such as a database, web server, or file server. In some cases, the server may be run on several nodes in order to improve performance or reliability. The server node is accessed by other nodes running **client** software. The clients connect to the server and send it requests to perform some action. The server runs the request on behalf of the client and returns a response. This is a simple structure that clearly divides which nodes in the distributed system are responsible for which tasks.

In order to request an action to be performed, the clients can use either message passing or remote procedure calls (RPC). In message passing, the clients sends a message which contains some kind of instructions of what task to perform, and the server returns a response which the client must parse. RPC is discussed in the following section.

24.3 Remote Procedure Calls

A Remote Procedure Call is a way for making an action performed by another node appear as if they were a simple function call. With RPC, the client software contains a *stub* method which is a special function which bundles up its arguments and transmits them to the server that will run the RPC call. The server then waits for these bundled requests to arrive, processes them, and then returns a bundle to the client with the result. Meanwhile, the client's stub method waits for the reply; when it is received, it unpacks the bundle and returns the results of the RPC as if it were an ordinary function call. In this way, the OS and the RPC library are able to completely mask all of the networking details from the application making the calls.

In order for RPCs to work, the server needs a way to advertise what functions it can run, and the clients need a way to find the servers. This process is called **naming**, and refers to both how the server registers its available functionality, and how the clients determine how to reach the servers to make calls. The servers define **signatures** of the functions they support, which define what input parameters they expect and what the return value type will be. Based on these signatures, a **stub compiler** builds a stub functions to be used by the clients and servers so they agree on how the method will be invoked.

The Java programming language supports RPC calls using the Remote Method Invocation (RMI) library. Many other languages support RPC through specialized libraries, and RPC is the most common model for communication between nodes in distributed systems. In fact, RPC can even be used within a single system in order to allow for simple inter-process communication.