CT414

Distributed Systems & Co-Operative Computing

Name: Andrew Hayes
Student ID: 21321503
E-mail: a.hayes18@universityofgalway.ie

2025–01–14

Contents

1 Introduction

1.1 Client-Server Architectures

1.1.1 Two-Tier Architectures

A **two-tier client-server architecture** is a client-server architecture wherein a client talks directly to a server, with no intervening server. It is typically used in small environments ($\lesssim$ 50 users).

A common development error is to prototype an application in a small, two-tier environment, and then scale up by simply adding more users to the server: this approach will usually result in an ineffective system, as the server becomes overwhelmed. To properly scale to hundreds or thousands of users, it is usually necessary to move to a three-tier architecture.

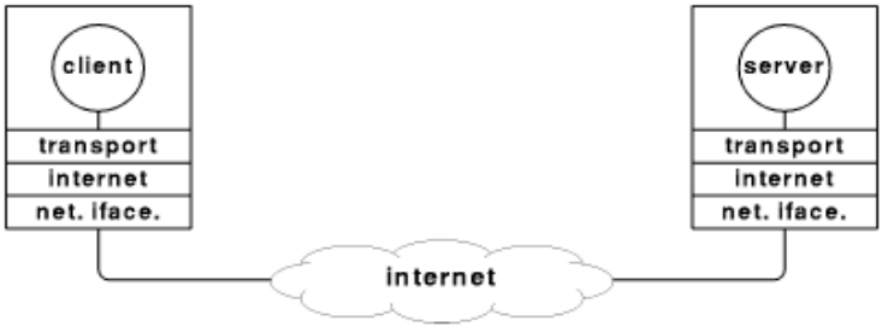

Figure 1: Client & server using TCP/IP protocols to communicate. Information can flow in either or both directions. The client & server can interact with a transport layer protocols.

1.2 Three-Tier Architecture

A **three-tier client-server architecture** introduces a server or **agent** (or **load-balancer**) between the client & the server. The agent has many roles:

- Translation services: such as adapting a legacy application on a mainframe to a client-server environment.
- Metering services: such as acting as a transaction monitor to limit the number of simultaneous requests to a given server.
- Intelligent agent services: as in mapping a request to a number of different servers, collating the results, and returning a single response to the client.

1.3 Network Programming Paradigms

Practically all network programming is based on a client-server model; the only real difference in paradigms is the **level** at which the programmer operates. The sockets API provides direct access to the available transport layer protocols. RPC is a higher-level abstraction that hides some of the lower-level complexities. Other approaches are also possible:

- Sockets are probably the best-known and most widely-used paradigm. However, problems of data incompatibility across platforms can arise.
- RPC libraries aim to solve some of the basic problems with sockets and provide a level of transport independence.
- Neither approach works very well with modern applications (Java RMI and other mdoern technologies, e.g., web services are better).

2 Java RMI

Remote Method Invocation (RMI) is a Java-based mechanism for distributed object computing. RMI enables the distribution of work to other Java objects residing in other processes or on other machines. The objects in one Java Virtual Machine (JVM) are allowed to seamlessly invoke methods on objects in a remote JVM. To call a method of a remote object, we must first get a reference to that object, which can be obtained from the registry name facility or by receiving the reference as an argument or return value of a method call. Clients can call a remote object in a server that itself is a client of another server. Parameters of method calls are passed as serialised objects:

- types are not truncated, and therefore, object-oriented polymorphism is supported;
- parameters are passed by value (deep copy) and therefore object behaviour can be passed.