1

THE PEER-TO-PEER ARCHITECTURE

In this chapter, we look at the general architecture of a peer-to-peer system and contrast it with the traditional client-server architecture that is ubiquitous in current computing systems. We then compare the relative merits and demerits of each of these approaches toward building a distributed system.

We begin the chapter with a discussion of the client-server and peer-to-peer computing architectures. The subsequent subsections look at the base components that go into making a peer-to-peer application, finally concluding with a section that compares the relative strengths and weaknesses of the two approaches.

1.1 DISTRIBUTED APPLICATIONS

A distributed application is an application that contains two or more software modules that are located on different computers. The software modules interact with each other over a communication network connecting the different computers.

To build a distributed application, you would need to decide how many software modules to include in the application, how to place those software modules on the different computers in the network, and how each software module discovers the other modules it needs to communicate with. There are many other tasks

Legitimate Applications of Peer-to-Peer Networks, by Dinesh C. Verma
ISBN 0-471-46369-8

that must be done to build a distributed application, but those mentioned above are the key tasks to explain the difference between client-server computing and peer-to-peer computing.

1.1.1 A Distributed Computing Example

The different approaches to distributed computing can be explained best by means of an example. Suppose you are given the task of creating a simulation of the movement of the Sun, the Earth, and the Moon by a team of five astronomers. Each of the five astronomers has a computer on which he or she would like to see the motion and position of the three heavenly bodies at any given time for the last 2000 years as well as the next 2000 years. Let us say (purely for the sake of illustration, rather than as the preferred way to write such simulation) that the best way to solve this problem is to create a large database of records, each record containing the relative positions of the three bodies at different times ranging over the entire 4000-year period. To show the positions of the three heavenly bodies, the program will find the appropriate set of records and display the position visually on the computer screen. Even after making this choice on how to write the program, you, as the programmer assigned to the task, have multiple ways to develop and deploy the software.

You can write a stand-alone program that will do the complete simulation of the three heavenly bodies that runs on a single computer and install five copies of it on each of the computers. This approach (approach I) has the advantage that each astronomer can run the program as long as his or her computer is up and does not require access to a network, or to the computers of the other astronomers. This approach would be fine if the application runs well enough on each of the computers, but it does not harness the combined processing power of all the five computers. Furthermore, experienced programmers know that all programs must be maintained and upgraded multiple times—to fix bugs, to add new features, or to correct any errors in generation of the set of records in the program. With this approach, any changes that you make after the initial installation of the program would need to be replicated five times.

An alternative approach (approach II) would be for you to select the most powerful computer among the five to do the simulation,

with all the other computers (as well as the one running the simulation) having a visualization interface for the users to interact with the simulator. You have broken the program into two software modules, the visualization module and the simulation module, and created five instances of the visualization module and one instance of the simulation module. If the astronomers' computers are of differing power, this allows all of them to harness the power of the fastest computer. Because the simulation module maintains a large set of data, this set can be maintained at a single place and use less disk space. Also, you can localize changes to the simulation module to a single computer, and only changes to the visualization module must be propagated to all of the five computers. If the visualization module is much simpler than the simulation module, this will cut down significantly on the number of bugs and changes that need to be maintained in different places. The drawback now is that each astronomer needs connectivity to the network in order to access the simulation module and that the computer running the simulation module must be available continuously.

Approach II outlined above follows the client-server architecture for distributed applications where the fastest computer is acting as the simulation server. The visualization modules running on the other computers are the clients that accesses the simulation module running on the server.

Although approach II allowed each computer to access the resources of the fastest computer, it did not use the combined processing power of the other four computers available to the distributed application. To use the processing power of all the five computers, you can divide your simulation modules into five identical portions, each one handling a different but similar part of the simulation process (approach III). Recalling the fact that the simulation module was implemented as a database of relative positions, each of the five computers can be assigned to hold a portion of the database. One could split the database into five equal portions, each computer holding one portion, or one could divide the database into overlapping portions so that the position at any time is stored at two or more computers. If disk space is not an issue, one could simply replicate the database on all the computers. When an astronomer wants to check the position of the three heavenly bodies at any time, the visualization module on his/her computer finds one of the five computers that has the simulation

module with the correct portion of the database and then talks to that simulation module. All the five computers are acting as peers, each having a client component (the visualization module) as well as a server component (the simulation module). Approach III is the pure peer-to-peer approach to solving the three-body simulation problem.

Approach III could potentially be more scalable than approach II because it is leveraging the combined power of all the computers rather than that of a single computer. If the records in the database are available from multiple computers, the reliability of this approach may be higher than that of approach II. However, it reintroduces the problem that any changes made to a module (simulation or visualization) need to be replicated on all of the different computers.

In real life, one could also use a hybrid approach that is a mixture between the client-server architecture and the peer-to-peer architecture. The hybrid approach places some software modules on a set of computers that can act as servers and others act as clients. The hybrid approach for some distributed applications can often result in a better trade-off between the ease of software maintenance, scalability, and reliability.

For any of the approaches selected, you would need to solve the discovery problem. The different modules of the application need to communicate with each other, and a prerequisite for this would be that the modules know where to send messages to the other modules. In the Internet, messages are sent to other applications by specifying their network address, which consists of the IP address of the application and the port numbers on which the application is receiving messages. To communicate over the Internet protocol suite, each software module must find out the network address of the other software module (or modules).

One solution to the discovery process is to fix the port numbers for all the software modules that they will be using and have all the modules know the port numbers and IP addresses of the different modules. When developing the simulation application for the astronomers, you can hard code this information within each of the modules. However, you must ensure that the selected port numbers are available on the computers that the applications will be running on. Because most computers run applications developed by many different companies, this solution would require a global coordination of port numbers among all the software devel-

opers in the world, which is clearly not feasible. The alternative is to have the address and port number information be provided as configuration parameters to the different software modules. If you use this approach with the example application we have discussed here, it is relatively easy to specify five port numbers and IP addresses in the configuration of each computer. However, if you consider the case of a more complex real-world application that needs to run on many more computers, the manual effort required for configuration could be quite substantial.

One of the key advantages of the client-server architecture (approach II discussed above) is that it makes the discovery process quite simple. This enables the deployment of a large number of clients and a high degree of scalability. Let us now define the client-server architecture and the peer-to-peer computing architecture in a more precise manner and then examine the discovery process in each of the architectures.

1.1.2 Client-Server Architecture

The client-server architecture is a way to structure a distributed application so that it consists of two distinct software modules:

- A server module, only one instance of which is present in the system
- A client module, of which multiple instances are present in the system

The only communication in the system is between the client modules and the server module.

Please note that the client and server modules themselves may be quite complex systems with further submodules and components. However, the key characteristic of the client-server architecture is that there is a server module that is the central point for communication. Clients do not communicate with each other, only with the server module.

In the client-server architecture, the server is usually the more complex piece of the software. The clients are often (although not always) simpler. With the wide availability of a web browser on most desktops, it is quite common to develop distributed applications so that they can use a standard web browser as the client. In

this case, no effort is needed to develop or maintain the client (or, rather, the effort has been taken over by a third party—the developer of the web browser). This simplifies the task of maintaining and upgrading the application software.

In any distributed application, the different components must discover each other in order to communicate. In the client-server architecture, only the clients need to communicate with the server. Therefore, each client needs to discover the network address of the server, and the server needs to know the network address of each of the clients.

The solution used for discovery in the client-server architecture is quite simple. The server runs on a port and network address that is known to the client module. The clients connect to the server on this well-known network address. Once the client connects to the server, the client and server are able to communicate with each other. The server need not be configured with any information about the clients. This implies that the same server module can communicate with any number of clients, constrained only by the physical resources needed to provide a reasonable response time to all of the connected clients.

For most common applications that run on the Internet, the port numbers on which the server side can run have been standardized [1]. Thus the clients only need to know the IP address of the computer on which the server is running. Any individual client can also easily switch to another server module by using the IP address (or, in general, the IP address and the port number) of the new server. As an example, a web server typically runs on port 80 and web browsers can connect to the web server when a user specifies the name of the computer running the web server. The browser also has the option of connecting to a server running on a port different than 80.

The simplicity and ease of maintenance of client-server architecture are the key reasons for its widespread usage in the design of distributed applications at the present time. However, the client-server architecture has one drawback—It does not utilize the computing power of the computers running the client modules as effectively as it does the computing power of the server module. At present, when even the standard desktop packs more computing power than the computers that were used for Neil Armstrong's flight to the Moon in 1969, this does appear to be a rather wasteful approach.

1.1.3 Peer-to-Peer Architecture

The peer-to-peer architecture is a way to structure a distributed application so that it consists of many identical software modules, each module running on a different computer. The different software modules communicate with each other to complete the processing required for the completion of the distributed application.

One could view the peer-to-peer architecture as placing a server module as well as a client module on each computer. Thus each computer can access services from the software modules on another computer, as well as providing services to the other computer. However, it also implies that the discovery process in the peer-to-peer architecture is much more complicated than that of the client-server architecture. Each computer would need to know the network addresses of the other computers running the distributed application, or at least of that subset of computers with which it may need to communicate. Furthermore, propagating changes to the different software modules on all the different computers would also be much harder. However, the combined processing power of several large computers could easily surpass the processing power available from even the best single computer, and the peer-to-peer architecture could thus result in much more scalable applications.

The bulk of this book is devoted to the subject of peer-to-peer applications. In Section 1.2, we look at the architecture of the typical software that must run on each computer in the peer-to-peer architecture. Subsequent chapters in the book discuss the issues of discovery and creating communication overlays among all the nodes that are participating in the peer-to-peer architecture.

1.2 THE PEER-TO-PEER SOFTWARE STRUCTURE

As mentioned above, a distributed application implemented in a peer-to-peer fashion would have the same software module running on all of the participating modules. Given the complexity associated with discovering, communicating, and managing the large number of computers involved in a distributed system, the software module is typically structured in a layered manner. The software of most peer-to-peer applications can be divided into

the three layers, the base overlay layer, the middleware layer, and the application layer.

The base overlay layer deals with the issue of discovering other participants in the peer-to-peer system and creating a mechanism for all the nodes to communicate with each other. This layer is responsible for ensuring that all the participants in the nodes are aware of the other participants. Functions provided by the base layer are the minimum functionality required for a peer-to-peer application to run.

The middleware layer includes additional software components that could be potentially reused by many different applications. The term "middleware" is used to refer to software components that are primarily invoked by other software components and used as a supporting infrastructure to build other applications. Functions included in this layer include the ability to create a distributed index for information in the system, providing a publish-subscribe facility, and security services. The functions provided in the middleware level are not necessary for all applications, but they are developed to be reused by more than one application.

Finally, the application layer provides software packages intended to be used by human users and developed so as to exploit the distributed nature of the peer-to-peer infrastructure.

Some implementation of the base functionality is needed in all peer-to-peer systems. Some peer-to-peer systems [2, 3] only provide middleware functionality and enable other applications to be written on top of them. Other peer-to-peer systems provide complete applications by using a common middleware [4] or by providing their own private implementation [5].

Please note that there is no standard terminology across different implementations of peer-to-peer systems, and thus the terms used above are general descriptions of the functionality needed for building a generic peer-to-peer system, rather than the structure of any specific peer-to-peer system. Also, most peer-to-peer systems are developed as single applications. However, the structuring of the three layers as discussed in this chapter provides a good way to categorize and study the different applications.

1.2.1 Base Overlay Layer

As mentioned above, the base overlay formation is a feature that must be provided by all peer-to-peer systems. The functions included in this layer include the following:

- *Discovery:* Before communicating with each other, a node in a peer-to-peer system must discover a minimum set of other nodes so that it could communicate with them. The discovery mechanism may include discovering all the other nodes in the system or just one other node that could be used as an intermediary to communicate with the other nodes.
- *Overlay Formation:* This provides a mechanism by which all the peer-to-peer nodes are connected into some type of common network. The network is used by each of the nodes to communicate with the other nodes.
- *Application-Level Multicast:* This functionality permits a node to send a message out to all of the other nodes in the network. In some peer-to-peer infrastructures, the only communication supported is the ability of a node to send a message to all of the other nodes. Some other peer-to-peer architectures would allow formation of subgroups within the system so that the message is sent only to a restricted subset of nodes.

1.2.2 Middleware Functions

The middleware layer is responsible for providing some common functions that will be used by applications at the higher layer. The middleware consists of those software functions that are intended to be used primarily by other software components, rather than by a human user. The middleware function in itself cannot be used to build a complete application, but the common functions can be used to build peer-to-peer applications rapidly.

Some of the functions included in this layer are:

- *Security:* This middleware function provides the support needed for managing secure communication among the different nodes, providing support such as encryption, access control, and authentication. The issues involved in the security aspects of peer-to-peer communication are similar to those involved in traditional client-server computing systems. However, the distributed nature of the system makes security issues much harder in peer-to-peer systems.
- *Distributed Indexing:* Many peer-to-peer applications need a fast way to index and find information that is distributed along the different nodes of a peer-to-peer infrastructure. A

distributed indexing system could be used by applications such as a distributed storage application or a distributed file system. A special type of an index is a hash table, which maps keywords of arbitrary length into a fixed-length hash and uses the hash to locate the entries corresponding to the keywords. Distributed indexing and hash tables have been an active area of research in peer-to-peer computing systems.

- *Directory Services:* A directory service provides a name lookup service, whereby one could look up the properties of an entity by specifying its name. In many respects, a directory service is like a index or hashing service, with one difference. It is common in conventional directory services to impose a hierarchical naming structure on the elements stored in the directory. The most widespread directory servers use the LDAP protocol to allow clients to access the information stored in the directory service. A peer-to-peer implementation of the directory service can offer some unique advantages over the traditional server-based implementation.
- *Publish-Subscribe Systems:* A publish-subscribe system allows for the sharing of information in a system in a controlled manner. Publishers of information send the information to the publish-subscribe system, and the subscribers of information inform the system about what types of information they wish to receive. The publish-subscribe system manages the published information sources and the preferences of the different subscribers and provides an efficient delivery mechanism.

Most middleware functions can be implemented with a client-server approach as well as a peer-to-peer approach. In the subsequent chapters of the book, we look at the different middleware functions that can be provided with a peer-to-peer infrastructure and compare the alternative implementations of the middleware functions.

1.2.3 Application Layer

We define this layer as consisting of the software components that are designed to be used primarily by a human user. The file-sharing application is the most ubiquitous peer-to-peer applica-

tion, with multiple implementations available from a large number of providers. The file-sharing application allows users of a peer-to-peer network to find files of interest from other computers on the network and to download them locally. The use of this application for sharing copyrighted content has been the subject of several legal cases between developers of peer-to-peer software and the music industry.

File sharing, however, is not the only application that can exploit the properties of a distributed base overlay infrastructure. A peer-to-peer infrastructure can be used to support self-managing websites, assist users to surf the network in an anonymous manner, and provide highly scalable instant messaging services and a host of other common applications.

There are some old applications which were built and developed with the peer-to-peer model long before the file-sharing application grew in prominence. These applications include some routing protocols used within the Internet infrastructure as well as the programs used to provide discussion and distributed newsgroups on the Internet.

Several such applications, new and legacy, are discussed in the subsequent chapters of this book. Each of these applications can be implemented with the client-server architecture or the peer-to-peer architecture. With each application, there are unique advantages and drawbacks in development using the peer-to-peer structure as compared to the client-server structure. However, some of these advantages and disadvantages pertain across all such applications and are discussed in Section 1.3.

1.3 COMPARISON OF ARCHITECTURES

If you had to implement an application and had the choice of implementing it with a peer-to-peer architecture or a client-server architecture, which one would you pick? Either of the two approaches to building the application can be made to work in most cases. In this section, we look at some of the issues you should consider when deciding which of the two approaches would be more appropriate for the task at hand. Each of the subsections discuss some of the issues you may want to consider and the merits and demerits of the two architectures compared with each other.

1.3.1 Ease of Development

When building an application, you need to consider how easy or difficult it will be to build and test the software for the application. The task of developing the software is eased by the existence of development and debugging tools that can be used to hasten the task of developing the application.

For developing client-server applications, there are a large number of application programs that are available to ease the task of development. Many software components, such as web servers, web-application servers, and messaging software, are available from several vendors and provide infrastructure that can be readily used to provide a server-centric solution.

Some programming environment packages are available for peer-to-peer computing, such as Sun's JXTA package [6] or Windows XP P2P SDK [7]. However, these packages are relatively new compared with the more traditional client-server software and therefore not quite as mature. Thus the risk of running into an undiscovered bug in the infrastructure is higher for peer-to-peer packages compared with those for client-server computing.

Furthermore, the task of debugging and testing a centralized server solution is easier than the task of debugging distributed software that requires interaction among several components. Thus, from an ease of development perspective, the client-server approach has an advantage over the peer-to-peer approach.

1.3.2 Manageability

Manageability refers to the ease of managing the final software when it is finally deployed. After a software application is up and ready, it still needs ongoing maintenance while in operation. Maintenance includes tasks such as ensuring that the application has not stopped working (and restarting it in case it stops working), making backup copies of the data generated by the application, applying software upgrades, fixing any bugs that are discovered, educating users about the application, and a variety of other functions.

Several tasks (e.g., user education) associated with managing a running application remain essentially unchanged regardless of the implementation approach that is used. However, many tasks (e.g., backup, upgrades, bug fixes) are easier to do on a centralized application than on an application that is run on multiple plat-

forms. An associated issue with peer-to-peer applications is the different number of platforms that the software would need to run on. For a centralized client-server approach, the server part of the software is at a single location and one could choose the platform on which the server part of the application would run. Choosing a platform in this instance means selecting the hardware and operating system of the machines on which the application software will run. A common platform allows for an improved degree of manageability on the server component of the software. The platform choices for the client component cannot be similarly restricted. However, for a client-server architecture that is developed with a standard client (e.g., a web browser), very little maintenance is associated with the client portion.

In a peer-to-peer application, the application is running on several different machines that could be distributed across a wide geographic area. If they are all under a single administrative domain, it is possible to standardize on a common platform for all of them. However, it is more common to find the situation in which the different components of a peer-to-peer application would run on different platforms. This makes the manageability of peer-to-peer applications much harder.

The advent of platform-independent programming languages such as Java has simplified the manageability of distributed peer-to-peer applications to a large extent. Furthermore, the dominance of the Windows computing platform on the desktop market has limited the number of potential platforms that a peer-to-peer application needs to run on. These factors help the manageability of peer-to-peer applications to a large extent. However, in general, a peer-to-peer application is typically less manageable than a client-server application.

1.3.3 Scalability

The scalability of an application is measured in terms of the highest rate or size of user-level interactions that the application can support with a reasonable performance. The quantity in which scalability is measured is determined by the type of application. The scalability of a web server can be measured by the number of user requests it can support per second; the scalability of a directory server can be measured by the number of operations it can support per second, as well as by the maximum number of records

it can store while maintaining a reasonable performance, for example, the maximum number of records it can store while keeping the time for a lookup operation below 100 ms.

Peer-to-peer applications use many computers to solve a problem and thus are likely to provide a more scalable solution than a server-centric solution, which relies on a single computer to perform the equivalent task. In general, using multiple computers would tend to improve the scalability of the application compared with using only a single computer.

However, a server-centric solution could be developed that uses multiple computers as well. Most high-performance server sites typically deploy many computers with a load balancer or dispatcher in front of the servers to provide a scalable solution. A dispatcher device distributes incoming requests to one of the many servers. Each server can process the request in the same manner. In most cases, the dispatcher maintains the affinity between clients and servers, that is, the dispatcher remembers which client requests were forwarded to which server and forwards multiple requests from the same client to the same server. An increase in the number of servers helps the system handle a large volume of requests. More details can be found in [8].

In general, if one uses the same number of computers to solve the problem, using all of the computers as servers would provide a more scalable system than using the same number of computers in a distributed peer-to-peer manner. This is because the peer-to-peer infrastructure requires communication among different nodes to perform the various tasks and thus has a larger overhead compared with a server-centric approach. In most cases, a centralized solution is much more efficient than a distributed solution.

The scalability of client-server computing as well as peer-to-peer computing has been proven by experience. Web servers of popular websites such as cnn.com or yahoo.com can handle millions of requests each day on a routine basis. Similarly, the large number of files exchanged on existing peer-to-peer networks such as gnutella and kazaa is on the order of millions of files every day. However, there is one difference between the scalability of the centralized server solution and the peer-to-peer solution. To build a server-centric solution, one would typically need to procure dedicated computers and host them at a facility. It is possible in many peer-to-peer applications to leverage existing computers

(e.g., workstations and laptops) that are not always heavily used. This enables a peer-to-peer infrastructure to harness many more computers at very little cost and thus allows a more scalable solution at a lower cost.

A fair criticism of the low-cost aspect of peer-to-peer systems such as gnutella or kazaa is that they are getting a free ride on the costly infrastructure paid for and supported by users and Internet service providers (ISPs). Although the criticism is valid, nothing prevents enterprise or industry operators to use the same trick to provide low-cost solutions to scalability. Most enterprises have hundreds of computers, desktops, and laptops, which are only using a fraction of their computational power and capacity. By harnessing their computing power with a peer-to-peer approach, enterprises can build scalable applications at a lower cost than that of comparable systems using dedicated servers.

1.3.4 Administrative Domains

One of the key factors determining how to structure the application would depend on the usage pattern of the application and how the different computers that are used to deploy the application software are going to be administered. In general, with a client-server approach, the server computers need to be under a single administrative domain. Thus the server-centric approach would typically not be used if the servers needed to host the application would belong to several different administrative domains.

A peer-to-peer system, however, can often be created by using computers from many different administrative domains. Thus, if usage of the software requires that computers from many different administrative domains be used, the peer-to-peer approach would be the natural choice for that application.

1.3.5 Security

Once an application has been deployed, one of the administrative tasks associated with it is to manage its security. Security management entails the tasks of making sure that the system is only accessed by authorized users, that user credentials are authenticated, and that malicious users of the system do not plant viruses or Trojan horses on the system.

Security issues and vulnerabilities have been studied compre-

hensively in server-centric solutions, and safeguards against the most common types of security attacks have been developed. In general, the security of a centralized system can be managed much more readily than the security of a distributed infrastructure. In a distributed infrastructure, the security apparatus and mechanisms must be replicated at multiple sites as opposed to a single site. This increases the cost of providing the security infrastructure. Furthermore, the existence of multiple sites allows for an increased vulnerability because there are more points that can be attacked by a hacker. In peer-to-peer applications that are written to run on computers across multiple administrative domains, the security issues are even harder to solve.

1.3.6 Reliability

The reliability of a system is measured by its ability to continue working when one or more of its components fail. In the context of computer systems, reliability is often measured in terms of the ability of the system to run when one or more computers hosting the system are brought down. The approach used for reliability in most computers is to provide for redundant components, having multiple computers do the task instead of a single computer, such as having standbys that can be activated when a primary computer fails.

High-reliability computer applications can be developed by using either client-server or peer-to-peer architectures. The solution for scalability for high-volume servers also provides for increased reliability and continued operation in case one of the servers fails. Distributed peer-to-peer systems, for most applications, use multiple computers to do identical tasks, and thus the system continues to be operational and available, even when a single computer fails or goes off-line. The most popular peer-to-peer networks are made up of thousands of computers. Although each computer in itself is a simple desktop and goes out of service frequently (when users switch off their machines), the entire system keeps on functioning without interruption.

As in the case of scalability, the difference between the reliability of a server-centric approach and the peer-to-peer approach is that of the cost at which the reliability is achieved. The peer-to-peer approach provides for a much more lower-cost solution for reliability than the server-centric approach.

Summarizing the overall discussion, we can say that a client-server approach provides for better security, manageability, and ease of development, whereas the peer-to-peer approach provides for increased reliability and scalability in a more cost-efficient manner and allows for interoperation across multiple administrative domains.