Chapter 16
Distributed Processing, Client/Server, and Clusters

Operating Systems: Internals and Design Principles
Eighth Edition
By William Stallings
### Table 16.1
**Client/Server Terminology**

<table>
<thead>
<tr>
<th><strong>Applications Programming Interface (API)</strong></th>
<th>A set of function and call programs that allow clients and servers to intercommunicate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client</strong></td>
<td>A networked information requester, usually a PC or workstation, that can query database and/or other information from a server</td>
</tr>
<tr>
<td><strong>Middleware</strong></td>
<td>A set of drivers, APIs, or other software that improves connectivity between a client application and a server</td>
</tr>
<tr>
<td><strong>Relational Database</strong></td>
<td>A database in which information access is limited to the selection of rows that satisfy all search criteria</td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>A computer, usually a high-powered workstation, a minicomputer, or a mainframe, that houses information for manipulation by networked clients</td>
</tr>
<tr>
<td><strong>Structured Query Language (SQL)</strong></td>
<td>A language developed by IBM and standardized by ANSI for addressing, creating, updating, or querying relational databases</td>
</tr>
</tbody>
</table>
**Client/Server Computing: Client**

- **Client** machines are generally single-user workstations providing a user-friendly interface to the end user.
  - client-based stations generally present the type of graphical interface that is most comfortable to users, including the use of windows and a mouse.
  - Microsoft Windows and Macintosh OS provide examples of such interfaces.
  - client-based applications are tailored for ease of use and include such familiar tools as the spreadsheet.
Each *server* provides a set of shared services to the clients

- most common type of server currently is the database server, usually controlling a relational database
- enables many clients to share access to the same database
- enables the use of a high-performance computer system to manage the database
Figure 16.1  Generic Client/Server Environment
A client/server configuration differs from other types of distributed processing:

- there is a heavy reliance on bringing user-friendly applications to the user on his or her own system
- there is an emphasis on centralizing corporate databases and many network management and utility functions
- there is a commitment, both by user organizations and vendors, to open and modular systems
- networking is fundamental to the operation
Figure 16.2 Generic Client/Server Architecture
Client/Server Applications

- The key feature of a client/server architecture is the allocation of application-level tasks between clients and servers.
- Hardware and the operating systems of client and server may differ.
- These lower-level differences are irrelevant as long as a client and server share the same communications protocols and support the same applications.
Client/Server Applications

- It is the communications software that enables client and server to interoperate
  - principal example is TCP/IP
- Actual functions performed by the application can be split up between client and server in a way that optimizes the use of resources
- The design of the user interface on the client machine is critical
  - there is heavy emphasis on providing a graphical user interface (GUI) that is easy to use, easy to learn, yet powerful and flexible
Figure 16.3 Client/Server Architecture for Database Applications
Figure 16.4 Client/Server Database Usage

(a) Desirable client/server use
Initial query

100,000 possible records

Next query

1,000 possible records

Final query

One record returned

(b) Misused client/server

Figure 16.4 Client/Server Database Usage
Four general classes are:

- Host-based processing
- Server-based processing
- Cooperative processing
- Client-based processing
Not true client/server computing

- Traditional mainframe environment in which all or virtually all of the processing is done on a central host
- Often the user interface is via a dumb terminal
- The user’s station is generally limited to the role of a terminal emulator
(b) Server-based processing

- Server does all the processing
- Client provides a graphical user interface
- Rationale behind configuration is that the user workstation is best suited to providing a user-friendly interface and that databases and applications can easily be maintained on central systems
- User gains the advantage of a better interface
Application processing is performed in an optimized fashion.

- Complex to set up and maintain
- Offers greater productivity and efficiency

(c) Cooperative processing
All application processing is done at the client
Data validation routines and other database logic functions are done at the server
Some of the more sophisticated database logic functions are housed on the client side
This architecture is perhaps the most common client/server approach in current use
It enables the user to employ applications tailored to local needs
Figure 16.6  Three-tier Client/Server Architecture
File Cache Consistency

- When a file server is used, performance of file I/O can be noticeably degraded relative to local file access because of the delays imposed by the network.
- File caches hold recently accessed file records.
- Because of the principle of locality, use of a local file cache should reduce the number of remote server accesses that must be made.
- The simplest approach to cache consistency is to use file locking techniques to prevent simultaneous access to a file by more than one client.
Figure 16.7 Distributed File Caching in Sprite
To achieve the true benefits of the client/server approach, developers must have a set of tools that provide a uniform means and style of access to system resources across all platforms. This would enable programmers to build applications that look and feel the same. Enable programmers to use the same method to access data regardless of the location of that data. The way to meet this requirement is by the use of standard programming interfaces and protocols that sit between the application (above) and communications software and operating system (below).
Figure 16.8 The Role of Middleware in Client/Server Architecture
Figure 16.9 Logical View of Middleware
Figure 16.10  Middleware Mechanisms

(a) Message-Oriented Middleware

(b) Remote Procedure Calls

(c) Object request broker
Figure 16.11 Basic Message-Passing Primitives
Reliability versus Unreliability

- Reliable message-passing guarantees delivery if possible
  - not necessary to let the sending process know that the message was delivered (but useful)
- If delivery fails, the sending process is notified of the failure
- At the other extreme, the message-passing facility may simply send the message out into the communications network but will report neither success nor failure
  - this alternative greatly reduces the complexity and processing and communications overhead of the message-passing facility
- For those applications that require confirmation that a message has been delivered, the applications themselves may use request and reply messages to satisfy the requirement
Blocking versus Nonblocking

Nonblocking
- process is not suspended as a result of issuing a Send or Receive
- efficient, flexible use of the message passing facility by processes
- difficult to test and debug programs that use these primitives
- irreproducible, timing-dependent sequences can create subtle and difficult problems

Blocking
- the alternative is to use blocking, or synchronous, primitives
- Send does not return control to the sending process until the message has been transmitted or until the message has been sent and an acknowledgment received
- Receive does not return control until a message has been placed in the allocated buffer
Remote Procedure Calls

- Allow programs on different machines to interact using simple procedure call/return semantics
- Used for access to remote services
- Widely accepted and common method for encapsulating communication in a distributed system

Standardized

- the communication code for an application can be generated automatically
- client and server modules can be moved among computers and operating systems with little modification and recoding
Local stub
RPC mechanism
Local response
Local procedure calls
Local application or operating system

Local stub
RPC mechanism
Remote procedure call
Remote procedure call
Remote server application

Local response
Local stub
RPC mechanism
Local response
Local procedure call

Figure 16.12 Remote Procedure Call Mechanism
Parameter Passing/Parameter Representation

- Passing a parameter by **value** is easy with RPC
- Passing by **reference** is more difficult
  - a unique system wide pointer is necessary
  - the overhead for this capability may not be worth the effort
- The representation/format of the parameter and message may be difficult if the programming languages differ between client and server
# Client/Server Binding

## Nonpersistent Binding
- A binding is formed when two applications have made a logical connection and are prepared to exchange commands and data.
- Nonpersistent binding means that a logical connection is established between the two processes at the time of the remote procedure call and that as soon as the values are returned, the connection is dismantled.
- The overhead involved in establishing connections makes nonpersistent binding inappropriate for remote procedures that are called frequently by the same caller.

## Persistent Binding
- A connection that is set up for a remote procedure call is sustained after the procedure return.
- The connection can then be used for future remote procedure calls.
- If a specified period of time passes with no activity on the connection, then the connection is terminated.
- For applications that make many repeated calls to remote procedures, persistent binding maintains the logical connection and allows a sequence of calls and returns to use the same connection.
Synchronous versus Asynchronous

Synchronous RPC
• behaves much like a subroutine call
• behavior is predictable
• however, it fails to exploit fully the parallelism inherent in distributed applications
• this limits the kind of interaction the distributed application can have, resulting in lower performance

Asynchronous RPC
• does not block the caller
• replies can be received as and when they are needed
• allow client execution to proceed locally in parallel with server invocation
Object-Oriented Mechanisms

- Clients and servers ship messages back and forth between objects
- A client that needs a service sends a request to an object broker
- The broker calls the appropriate object and passes along any relevant data
- The remote object services the request and replies to the broker, which returns the response to the client
- The success of the object-oriented approach depends on standardization of the object mechanism
- Examples include Microsoft’s COM and CORBA
Clusters

- Alternative to symmetric multiprocessing (SMP) as an approach to providing high performance and high availability

- Group of interconnected, whole computers working together as a unified computing resource that can create the illusion of being one machine

  *Whole computer* means a system that can run on its own, apart from the cluster

- Each computer in a cluster is referred to as a **node**
Benefits of Clusters

Absolute scalability
- A cluster can have dozens or even hundreds of machines, each of which is a multiprocessor

Incremental scalability
- Configured in such a way that it is possible to add new systems to the cluster in small increments

High availability
- Failure of one node is not critical to system

Superior price/performance
- By using commodity building blocks, it is possible to put together a cluster at a much lower cost than a single large machine
Figure 16.13  Cluster Configurations

(a) Standby server with no shared disk

(b) Shared disk
<table>
<thead>
<tr>
<th>Clustering Method</th>
<th>Description</th>
<th>Benefits</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Passive Standby</strong></td>
<td>A secondary server takes over in case of primary server failure.</td>
<td>Easy to implement.</td>
<td>High cost because the secondary server is unavailable for other processing tasks.</td>
</tr>
<tr>
<td><strong>Active Secondary</strong></td>
<td>The secondary server is also used for processing tasks.</td>
<td>Reduced cost because secondary servers can be used for processing.</td>
<td>Increased complexity.</td>
</tr>
<tr>
<td><strong>Separate Servers</strong></td>
<td>Separate servers have their own disks. Data is continuously copied from primary to secondary server.</td>
<td>High availability.</td>
<td>High network and server overhead due to copying operations.</td>
</tr>
<tr>
<td><strong>Servers Connected to Disks</strong></td>
<td>Servers are cabled to the same disks, but each server owns its disks. If one server fails, its disks are taken over by the other server.</td>
<td>Reduced network and server overhead due to elimination of copying operations.</td>
<td>Usually requires disk mirroring or RAID technology to compensate for risk of disk failure.</td>
</tr>
<tr>
<td><strong>Servers Share Disks</strong></td>
<td>Multiple servers simultaneously share access to disks.</td>
<td>Low network and server overhead.</td>
<td>Requires lock manager software. Usually used with disk mirroring or RAID technology.</td>
</tr>
</tbody>
</table>
Operating System Design Issues

Failure Management

- Two approaches can be taken to deal with failures:

  Highly available clusters
  - offers a high probability that all resources will be in service
  - any lost query, if retried, will be serviced by a different computer in the cluster
  - the cluster operating system makes no guarantee about the state of partially executed transactions
  - if a failure occurs, the queries in progress are lost

  Fault-tolerant clusters
  - achieved by the use of redundant shared disks and mechanisms for backing out uncommitted transactions and committing completed transactions
  - ensures that all resources are always available
The function of switching an application and data resources over from a failed system to an alternative system in the cluster is referred to as **fallover**.

The restoration of applications and data resources to the original system once it has been fixed is referred to as **fallback**.

Fallback can be automated but this is desirable only if the problem is truly fixed and unlikely to recur.

Automatic failback can cause subsequently failed resources to bounce back and forth between computers, resulting in performance and recovery problems.
Load Balancing

- A cluster requires an effective capability for balancing the load among available computers
- This includes the requirement that the cluster be incrementally scalable
- When a new computer is added to the cluster, the load-balancing facility should automatically include this computer in scheduling applications
- Middleware must recognize that services can appear on different members of the cluster and may migrate from one member to another
Parallelizing Computation

**Parallelizing compiler**
- determines, at compile time, which parts of an application can be executed in parallel
- performance depends on the nature of the problem and how well the compiler is designed

**Parallelized application**
- the programmer writes the application from the outset to run on a cluster and uses message passing to move data, as required, between cluster nodes
- this places a high burden on the programmer but may be the best approach for exploiting clusters for some applications

**Parametric computing**
- this approach can be used if the essence of the application is an algorithm or program that must be executed a large number of times, each time with a different set of starting conditions or parameters
- for this approach to be effective, parametric processing tools are needed to organize, run, and manage the jobs in an orderly manner
Figure 16.14   Cluster Computer Architecture [BUYY99a]
Clusters Compared to SMP

- Both clusters and SMP provide a configuration with multiple processors to support high-demand applications
- Both solutions are commercially available
- SMP has been around longer
- SMP is easier to manage and configure
- SMP takes up less physical space and draws less power than a comparable cluster
- SMP products are well established and stable
- Clusters are better for incremental and absolute scalability
- Clusters are superior in terms of availability
Windows Cluster Server

Windows Failover Clustering is a shared-nothing cluster in which each disk volume and other resources are owned by a single system at a time.

The Windows cluster design makes use of the following concepts:

- **cluster service** – the collection of software on each node that manages all cluster-specific activity
- **resource** – an item managed by the cluster service
- **online** – a resource is said to be online at a node when it is providing service on that specific node
- **group** – a collection of resources managed as a single unit
Group

- Combines resources into larger units that are easily managed
  - both for failover and load balancing
- Operations performed on a group automatically affect all of the resources in that group
- Resources are implemented as DLLs
  - managed by a resource monitor
- Resource monitor interacts with the cluster service via remote procedure calls and responds to cluster service commands to configure and move resource groups
Cluster Management Tools

Cluster API DLL

RPC

Cluster Service

Database Manager

Global Update Manager

Node Manager

Event Processor

Communication Manager

Failover Mgr

Resource Mgr

Resource Monitors

Physical Resource DLL

Logical Resource DLL

App Resource DLL

Cluster-aware App

Other Nodes

App Resource DLL

Non-aware App

Figure 16.15 Windows Cluster Server Block Diagram [SHOR97]
- **Node Manager**: responsible for maintaining this node’s membership in the cluster
- **Configuration Database Manager**: maintains the cluster configuration database
- **Resource Manager/Failover Manager**: makes all decisions regarding resource groups and initiates appropriate actions
- **Event Processor**: connects all of the components of the cluster service, handles common operations, and controls cluster service initialization
Beowulf and Linux Clusters

Beowulf project:

- was initiated in 1994 under the sponsorship of the NASA High Performance Computing and Communications (HPPC) project
- goal was to investigate the potential of clustered PCs for performing important computation tasks beyond the capabilities of contemporary workstations at minimum cost
- is widely implemented and is perhaps the most important cluster technology available
Beowulf Features

- Mass market commodity items
- Dedicated processors and network
- A dedicated, private network
- No custom components
- Easy replication from multiple vendors
- Scalable I/O
- A freely available software base
- Use of freely available distribution computing tools with minimal changes
- Return of the design and improvements to the community
Ethernet or Interconected Ethernets

Figure 16.16 Generic Beowulf Configuration

Distributed shared storage

Linux workstations
Beowulf System Software

- Is implemented as an add-on to commercially available, royalty-free base Linux distributions
- Each node in the Beowulf cluster runs its own copy of the Linux kernel and can function as an autonomous Linux system
- Examples of Beowulf system software:
  - Beowulf distributed process space (BPROC)
  - Beowulf Ethernet Channel Bonding
  - Pvmsync
  - EnFuzion
Summary

- Client/server computing
  - What is client/server computing?
  - Client/server applications
  - Middleware
- Distributed message passing
  - Reliability versus unreliability
  - Blocking versus nonblocking
- Windows cluster server
- Beowulf and Linux clusters
  - Beowulf features
  - Beowulf software
- Remote procedure calls
  - Parameter passing
  - Parameter representation
  - Client/server binding
  - Synchronous versus asynchronous
  - Object-oriented mechanisms
- Clusters
  - Cluster configurations
  - Operating system design issues
  - Cluster computer architecture
  - Clusters compared to SMP