Two and Three tier architecture

The Tier

Definition:

A *tier* is a distinct part of *hardware* or *software.*

Discussion:

The most common *tier* systems are:

- Single Tier
- Two Tier
- Three Tier

Each are defined as follows:

Single Tier

Definition:

A single computer that contains a *database* and a *front-end* to access the database.

Discussion:

Generally this type of system is found in small businesses. There is one computer which stores all of the company's data on a single database. The *interface* used to interact with the database may be part of the database or another program which ties into the database itself.

Advantages*;*

A *single-tier* system requires only one stand-alone computer. It also requires only one installation of proprietary software. This makes it the most cost-effective system available.

Disadvantages:

My be used by only one user at a time. A single tier system is impractical for an organization which requires two or more users to interact with the organizational data store at the same time.

Client/Server

Definition:

A *client* is defined as a requester of services and a *server* is defined as the provider of services. A single machine can be both a client and a server depending on the software configuration.

File Sharing Architecture

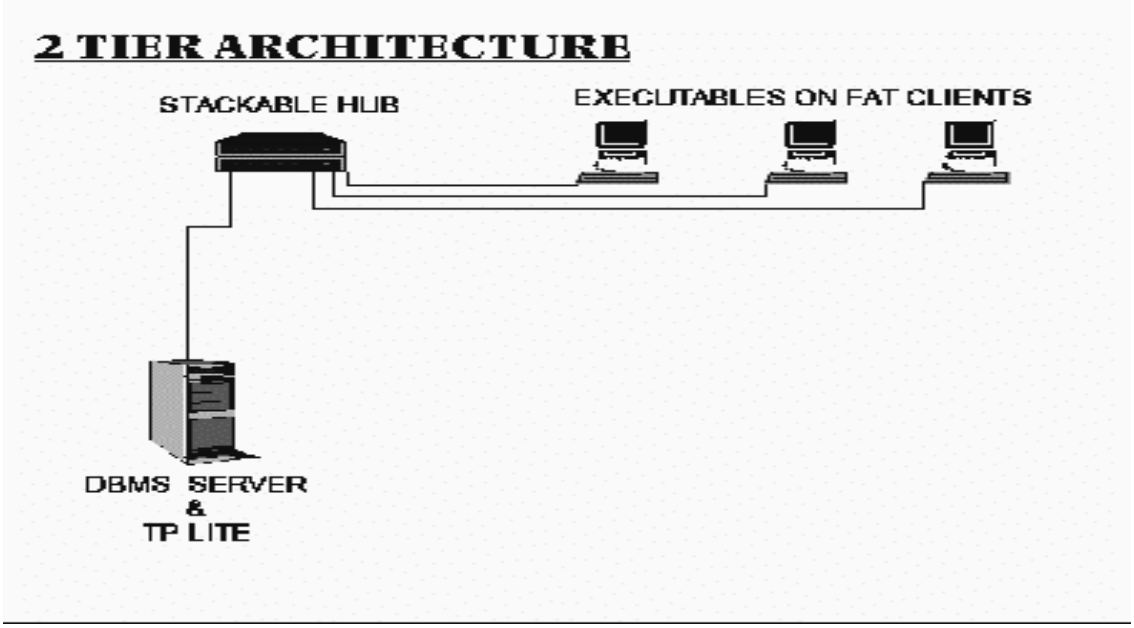
Definition:

Files used by the clients are stored on the server. When files are downloaded to the client all of the processing is done by the client. This processing includes all logic and data.

Discussion

File sharing architectures work if shared usage is low, update contention is low, and the volume of data to be transferred is low. The system gets strained when there are more than 12 users. This system was replaced by the *two tier client/server* architecture.

Two Tier Systems



Definition:

A *two tier* system consists of a *client* and a *server*. In a two tier system, the database is stored on the *server*, and the interface used to access the database is installed on the *client*.

Discussion:

The user system interface is usually located in the user's desktop environment and the database management services are usually in a server that is a more powerful machine that services many clients. Processing management is split between the user system interface environment and the database management server environment. The database management server provides stored procedures and triggers.

Purpose and Origin

Two tier software architectures were developed in the 1980s from the file server software architecture design. The two tier architecture is intended to improve *usability* by supporting a forms-based, user-friendly interface. The two tier architecture improves *scalability* by accommodating up to 100 users (file server architectures only accommodate a dozen users), and improves *flexibility* by allowing data to be shared, usually within a homogeneous environment The two tier architecture requires minimal

operator intervention, and is frequently used in non-complex, non-time critical information processing systems. Detailed readings on two tier architectures can be found in Schussel and Edelstein.

Technical Details

Two tier architectures consist of three components distributed in two layers: client (requester of services) and server (provider of services). The three components are

1. User System Interface (such as session, text input, dialog, and display management services)
2. Processing Management (such as process development, process enactment, process monitoring, and process resource services)
3. Database Management (such as data and file services)

The two tier design allocates the user system interface exclusively to the client. It places database management on the server and splits the processing management between client and server, creating two layers.

Usage Considerations

Two tier software architectures are used extensively in non-time critical information processing where management and operations of the system are not complex. This design is used frequently in decision support systems where the transaction load is light. Two tier software architectures require minimal operator intervention. The two tier architecture works well in relatively homogeneous environments with processing rules (business rules) that do not change very often and when workgroup size is expected to be fewer than 100 users, such as in small businesses.

Advantages:

Since processing was shared between the client and server, more users could interact with such a system.

Disadvantages:

When the number of users exceeds 100, performance begins to deteriorate. This limitation is a result of the server maintaining a connection via "keep-alive" messages with each client, even when no work is being done. A second limitation of the two tier architecture is that implementation of processing management services using vendor proprietary database procedures restricts flexibility and choice of DBMS for applications. Finally, current implementations of the two tier architecture provide limited flexibility in moving (repartitioning) program functionality from one server to another without manually regenerating procedural code.

Three Tier Architecture

Purpose and Origin

The three tier software architecture (a.k.a. three layer architectures) emerged in the 1990s to overcome the limitations of the two tier architecture (see Two Tier Software Architectures). The third tier (middle tier server) is between the user interface (client) and the data management (server) components. This middle tier provides process management where business logic and rules are executed and can accommodate hundreds of users (as compared to only 100 users with the two tier architecture) by providing functions such as queuing, application execution, and database staging. The three tier architecture is used when an effective distributed client/server design is needed that provides (when compared to the two tier) increased *performance*, *flexibility*, *maintainability*, *reusability*, and *scalability*, while hiding the complexity of distributed processing from the user

Technical Details

A three tier distributed client/server architecture (as shown in Figure 28) includes a user system interface top tier where user services (such as session, text input, dialog, and display management) reside.

The third tier provides database management functionality and is dedicated to data and file services that can be optimized without using any proprietary database management

system languages. The data management component ensures that the data is consistent throughout the distributed environment through the use of features such as data locking, consistency, and replication. It should be noted that connectivity between tiers can be dynamically changed depending upon the user's request for data and services.

The middle tier provides process management services (such as process development, process enactment, process monitoring, and process resourcing) that are shared by multiple applications.

The middle tier server (also referred to as the application server) improves performance, flexibility, maintainability, reusability, and scalability by centralizing process logic. Centralized process logic makes administration and change management easier by localizing system functionality so that changes must only be written once and placed on the middle tier server to be available throughout the systems.

Usage Considerations

The middle tier manages distributed database integrity by the two phase commit process. It provides access to resources based on names instead of locations, and thereby improves scalability and flexibility as system components are added or move.

Sometimes, the middle tier is divided in two or more unit with different functions, in these cases the architecture is often referred as multi layer. This is the case, for example, of some Internet applications. These applications typically have light clients written in HTML and application servers written in C++ or Java, the gap between these two layers is too big to link them together. Instead, there is an intermediate layer (web server) implemented in a scripting language. This layer receives requests from the Internet clients and generates html using the services provided by the business layer. This additional layer provides further isolation between the application layout and the application logic.

It should be noted that recently, mainframes have been combined as servers in distributed architectures to provide massive storage and improve security.

Definition:

The addition of a middle tier between the user system interface client environment and the database management server environment.

Discussion:

There are a variety of ways of implementing this middle tier, such as transaction processing monitors, message servers, or application servers. The middle tier can perform queuing, application execution, and database staging.

Example:

If the middle tier provides queuing, the client can deliver its request to the middle layer and disengage because the middle tier will access the data and return the answer to the client. In addition the middle layer adds scheduling and prioritization for work in progress.

Advantages:

The three tier client/server architecture has been shown to improve performance for groups with a large number of users (in the thousands) and improves flexibility when compared to the two tier approach. modules onto different computers in some three tier architectures.

Disadvantages:

The three tier architectures development environment is reportedly more difficult to use than the visually-oriented development of two tier systems.

Ecommerce Systems - Application Servers

Definition:

Application servers share business logic, computations, and a data retrieval engine on the server. There is now processing required on the client.

Advantages:

With less software on the client there is less security to worry about, applications are more scalable, and support and installation costs are less on a single server than maintaining each on a desktop client. The application server design should be used when security, scalability, and cost are major considerations.

Multi-Tier Application Design

An age-old software engineering principle explains that by logically partitioning a piece of software into independent layers of responsibility, one can produce programs that have fewer defects, are better at documenting themselves, can be developed concurrently by may programmers with specific skill sets, and are more maintainable than the alternative of a monolithic hunk of code. Examples of these layers, or tiers, are common: the kernel (privileged CPU mode) and other applications (user mode); the seven ISO/OSI network model layers (or the redivided four used by the Internet); and even the database "onion" containing core, management system, query engine, procedural language engine, and connection interface.

Note that these tiers are entirely logical in nature. Their physical implementation may vary considerably: everything compiled into one EXE, a single tier spread across multiple statically- or dynamically-linked libraries, tiers divided amongst separate networked computers, and so forth.

Each such tier is one further level of abstraction from the raw data of the application (the "lowest" tier). The "highest" tier is therefore the most "abstract" and also the best candidate for communicating directly with the end user.

Individual tiers are designed to be a self-contained as possible, exposing only a well-defined interface (e.g. function names, usually called an Application Programming Interface, or API) that another tier may use. In this respect, tiers are analogous to the classes of Object-Oriented Programming. In theory, a new tier with a compatible interface could easily be substituted for another, but in practice this can't always be done without a bit of fuss.

Tiers only communicate in this downward direction (that is, a lower-level tier does not call a function in a higher-level tier), and a tier may only call into the tier directly beneath

it (that is, tiers are not bypassed). One might also say that a higher-level tier is a "consumer" of the services afforded by the lower-level tier, the "provider".

Each tier does introduce a small performance penalty (typically, stack frame overhead for the function calls) but this is usually not significant, and is outweighed by the design advantages of a multi-tier design.

If performance does become an issue, some of the rules above may be broken, with a consequent loss of design consistency.

Three tier VS Muti tier

These three tiers have proved more successful than other multi-tier schemes for a couple of reasons:

Matching Skill sets: It turns out that the skills of the various people that might work on building an application tend to correspond neatly with the three tiers. The Presentation tier requires people with some level of either user-interface/ergonomics experience or artistic sensibilities, or the combination of the two (often found in web designers). The Business Logic tier calls upon people very familiar with procedural language techniques and a considerable investment in a particular set of procedural programming languages (e.g. C/C++, Java, VB, PHP). Finally, the Data tier requires intimate knowledge of relational database theory and the SQL DML language. It's next to impossible to find a single person with talent and experience in all three areas, and reasonably difficult to find people with skills in two. Fortunately, the separation of the three layers means that people with just one of these skill sets can work on the project side by side with those possessing the other skills, lessening the "too many cooks" effect.

Multi-Server Scalability: Just as people become "optimized" for a certain task through learning and experience, computer systems can also be optimized for each of the three tiers. While it's possible to run all three logical tiers on a single server (as is done on the course server), as a system grows to accommodate greater and greater numbers of users (a problem typical of web applications), a single server will no longer suffice. It turns out that the processing needs of the three tiers are distinct, and so a physical arrangement often consists of many Presentation tier servers (a.k.a. web servers), a few Business

Logic tier servers (a.k.a. application servers), and usually just one, or at most a handful, of Data tier servers (a.k.a. RDBMS servers). RDBMS servers consume every resource a hardware platform can provide: CPU, memory, disk, and gobs of each. RDBMS's also often have innumerable tuning parameters. An application server is typically only CPU and memory-bound, requiring very little disk space. Finally, a web server (just a specialized type of file server) is mostly reliant on memory and disk.

Review Questions

1. Explain what is single tier database architecture?
2. Explain the Two-Tier architecture?
3. Explain the n tier architecture?

References

http://otn.oracle.com/pub/articles/tech_dba.html

http://www.openlinksw.com/licenses/appst.htmDate, C. J. *An Introduction to Database Systems*. Volume I. Addison/Wesley, 1990, 455–473.