Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

Bases de datos distribuidas

Equipo 3

Arquitectura DBMS distribuida

Arquitectura DBMS distribuida

“La arquitectura de un sistema define su estructura. Esto significa que los componentes están identificados, cada componente tiene su función especifica y las interrelaciones e interacciones entre estos componentes están definidas”

El texto cita tres tipos de arquitecturas “ideales” para un manejador de base de datos distribuidos:

- Un sistema cliente/servidor
- Un sistema punto a punto
- Un sistema con múltiples bases de datos

Lo de “ideal” va en base de que distintas arquitecturas comerciales ya implementadas toman su diseño más elemental de las arquitecturas citadas anteriormente.

La primera arquitectura que el texto aborda es la “ANSI/SPARC”, cuyo enfoque es las diferentes clases y roles de usuario, así como su respectiva vista de la información contenida.

1.7.1 Arquitectura ANSI/SPARC

En 1977 un reporte emitido por un grupo de estudio de Sistemas de administración de Bases de Datos del Comité de Planeación y requerimientos de Estándares bajo el Comité de computadoras y procesamiento de información del Instituto Americano de Estándares Nacionales con la misión de determinar si establecer estándares en dicha área era factible.

El grupo de estudio determino que las interfaces fueran estandarizadas, y que el framework de la arquitectura constara de 43 interfaces, catorce de las cuales serían la parte física del sistema de almacenamiento de la computadora por lo que no sería considerada esencial para la arquitectura del DBMS.

Si simplificamos la arquitectura se puede observar que existen tres vistas de los datos:

- Vista externa (para el usuario final)
- Vista interna (esta es la vista para el sistema o computadora)
- Vista conceptual (O la que es destinada para la empresa)

Además de esas vistas se tienen distintos niveles de la arquitectura, “el nivel conceptual con toda certeza será relacional, los objetos visibles serán tablas relacionales (los operadores serán también relacionales).

EL NIVEL externo casi siempre será relacional.

EL NIVEL interno no será “relacional” porque los objetos en ese nivel no serán por lo regular sólo tablas relacionales (almacenadas), sino que serán objetos similares a los encontrados en el nivel interno de otros tipos de sistemas.”

1.7.2 Arquitectura genérica

Cuando un DBMS se ejecuta en una computadora ordinaria se comunica con dos componentes, el subsistema de comunicación y el sistema operativo. El subsistema de comunicación permite enlazar el DBMS con otros subsistemas del equipo para permitir el enlace con aplicaciones. Un ejemplo claro es el uso de una terminal para comunicarse con el DBMS y poder ejecutar transacciones. Y de forma obvia el sistema operativo brinda el enlace entre el DBMS y los recursos físicos de la computadora.

Se determina que las funciones de un DBMS van de arriba hacia abajo, con capas de interfaz, control, compilación, ejecución, acceso de datos y administración de consistencia.

La capa de interfaz administra el enlace a las aplicaciones

La capa de control controla el query a realizar añadiendo la semántica y autorización necesaria.

La capa de compilación organiza los querys en una secuencia de operaciones de bajo nivel.

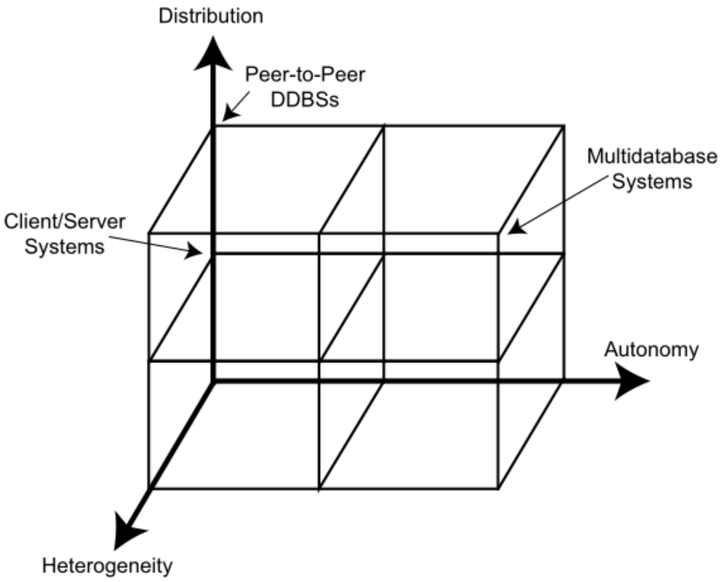
La capa de ejecución interpreta las operaciones relacionales mediante las solicitudes de búsqueda y actualización.

La capa de acceso de datos administra las estructuras de datos a implementar.

La capa de consistencia administra el control de concurrencia y los logs generados.

1.7.3 Modelo arquitectónico por las DBMS distribuidas

El modelo arquitectónico usa una clasificación que organiza los sistemas de acuerdo a su autonomía de los sistemas locales, su distribución y su heterogeneidad.



1.7.3 Autonomía

En la cultura popular, divide y vencerás hace referencia a un refrán que implica resolver un problema difícil. La autonomía se refiera a la distribución del control mas no el de los datos.

Desde la perspectiva de los usuarios, todos los datos están almacenados en un mismo sitio.

Las dimensiones de la autonomía se pueden especificar de la siguiente manera

1. Autonomía de diseño: los DBMS individuales son libres de usar los modelos de datos y las técnicas de gestión de transacciones que prefieran.

2. Autonomía de comunicación: Cada uno de los DBMS individuales es libre de tomar su propia decisión sobre qué tipo de información quiere proporcionar a los otros DBMS o al software que controla su ejecución global.

3. Autonomía de ejecución: Cada SGBD puede ejecutar las transacciones que se le sometan de la forma que desee.

1.7.4 Distribución

La dimensión de la distribución si se ocupa de los datos y difiere a lo que hace la autonomía.

La distribución cliente/servidor concentran las funciones de gestión de datos en los servidores, las tareas de comunicación se comparten entre maquinas clientes y los servidores.

En los sistemas peer-to-peer no existe la distinción entre maquina cliente y servidor., pues cada maquina tiene una funcionalidad DBMS completa y pueden comunicarse con otras maquinas para ejecutar consultas y transacciones.

1.7.5 Heterogeneidad.

Como consecuencia de la autonomía puede existir heterogeneidad, ya que se debe a que los datos de cada base de datos cuentan con diferentes tipos o formatos.

Estas bases de datos están en diferentes sitios utilizando gestores de bases de datos, siendo cada uno autónomo y pueden que utilicen esquemas y software de gestión de sistemas de bases de datos diferentes.

1.7.6 Alternativas arquitectónicas.

Existen principalmente tres tipos de arquitectura en DBMS.

1.- Arquitectura de una capa

Consiste en que el cliente, el servidor y la base de datos residen en la misma maquina.

2.- Arquitectura de dos capas.

La capa de presentación se ejecuta en un cliente (PC, Tablet, celular, etc.) y los datos son almacenados en un servidor llamado segunda capa.

Esta arquitectura nos proporciona seguridad adicional para el DBMS porque no esta expuesto directamente al usuario final. También proporciona una comunicación directa y mas rápida.

3.- Arquitectura de tres capas.

Incluimos una capa de presentación, una capa de aplicación y un servidor de bases de datos.

Son una extensión de la arquitectura de dos capas en la que los procesos funcionales, la lógica, el acceso a los datos, el almacenamiento da datos y la interfaz de usuario se desarrollan y mantienen de forma independiente como módulos separados.

1.7.8 Client/Server Systems

distinguish the functionality that needs to be provided and divide these functions into two classes: server functions and client functions. This provides a two-level architecture which makes it easier to manage the complexity of modern DBMSs and the complexity of distribution.

We focus on what software should run on the client machines and what software should run on the server machine.

In relational systems, the server does most of the data management work. This means that all of query processing and optimization, transaction management and storage management is done at the server. The client, in addition to the application and the user interface, has a DBMS client module that is responsible for managing the data that is cached to the client and (sometimes) managing the transaction locks that may have been cached as well.

Types of client/server architecture The simplest is the case where there is only one server which is accessed by multiple clients. We call this multiple client/single server. From a data management perspective, this is not much different from centralized databases since the database is stored on only one machine (the server) that also hosts the software to manage it. However, there are some (important) differences from centralized systems in the way transactions are executed and caches are managed.

Client/server can be naturally extended to provide for a more efficient function distribution on different kinds of servers: client servers run the user interface (e.g., web servers), application servers run application programs, and database servers run database management functions. This leads to the present trend in three-tier distributed system architecture, where sites are organized as specialized servers rather than as general-purpose computers.

The database server approach, as an extension of the classical client/server architecture, has several potential advantages. First, the single focus on data management makes possible the development of specific techniques for increasing data reliability and availability, e.g. using parallelism. Second, the overall performance of database management can be significantly enhanced by the tight integration of the database system and a dedicated database operating system. Finally, a database server can also exploit recent hardware architectures, such as multiprocessors or clusters of PC servers to enhance both performance and data availability.

The application server approach (indeed, a n-tier distributed approach) can be extended by the introduction of multiple database servers and multiple application servers, as can be done in classical client/server architectures.

1.7.9 Peer-to-Peer Systems

Massive distribution in current systems. While in the early days we focused on a few (perhaps at most tens of) sites, current systems consider thousands of sites. The second is the inherent heterogeneity of every aspect of the sites and their autonomy. While this has always been a concern of distributed databases, as discussed earlier, coupled with massive distribution, site heterogeneity and autonomy take on an added significance, disallowing some of the approaches from consideration.

Let us start the description of the architecture by looking at the data organizational view. We first note that the physical data organization on each machine may be, and probably is, different. This means that there needs to be an individual internal schema definition at each site, which we call the local internal schema (LIS). The enterprise view of the data is described by the global conceptual schema (GCS), which is global because it describes the logical structure of the data at all the sites.

To handle data fragmentation and replication, the logical organization of data at each site needs to be described. Therefore, there needs to be a third layer in the architecture, the local conceptual schema (LCS). In the architectural model we have chosen, then, the global conceptual schema is the union of the local conceptual schemas. Finally, user applications and user access to the database is supported by external schemas (ESs), defined as being above the global conceptual schema.

The first DBMS technologies that emerged in the 1960s supported hierarchical databases, in which data is organized in a tree-like structure with parent and child records, and network databases, which allowed relationships between databases to be mapped. data elements in different groupings of parents and children.

Data independence is supported since the model is an extension of ANSI/SPARC, which provides such independence naturally. Location and replication transparencies are supported by the definition of the local and global conceptual schemas and the mapping in between. Network transparency, on the other hand, is supported by the definition of the global conceptual schema.

The user queries data irrespective of its location or of which local component of the distributed database system will service it. As mentioned before, the distributed DBMS translates global queries into a group of local queries, which are executed by distributed DBMS components at different sites that communicate with one another.

Components of a distributed DBMS.

* User Processor

1. The user interface handler is responsible for interpreting user commands as they come in, and formatting the result data as it is sent to the user.

2. The semantic data controller uses the integrity constraints and authorizations that are defined as part of the global conceptual schema to check if the user query can be processed.

3. The global query optimizer and decomposer determines an execution strategy to minimize a cost function, and translates the global queries into local ones using the global and local conceptual schemas as well as the global directory. The global query optimizer is responsible, among other things, for generating the best strategy to execute distributed join operations.

4. The distributed execution monitor coordinates the distributed execution of the user request. The execution monitor is also called the distributed transaction manager.

* Data Processor

1. The local query optimizer, which actually acts as the access path selector, is responsible for choosing the best access path to access any data item

2. The local recovery manager is responsible for making sure that the local database remains consistent even when failures occur

3. The run-time support processor physically accesses the database according to the physical commands in the schedule generated by the query optimizer. The run-time support processor is the interface to the operating system and contains the database buffer (or cache) manager, which is responsible for maintaining the main memory buffers and managing the data accesses.

It is important to note, at this point, that our use of the terms “user processor” and “data processor” does not imply a functional division similar to client/server systems. These divisions are merely organizational and there is no suggestion that they should be placed on different machines. In peer-to-peer systems, one expects to find both the user processor modules and the data processor modules on each machine. However, there have been suggestions to separate “query-only sites” in a system from full-functionality ones. In this case, the former sites would only need to have the user processor.

In client/server systems where there is a single server, the client has the user interface manager while the server has all of the data processor functionality as well as semantic data controller; there is no need for the global query optimizer or the global execution monitor. If there are multiple servers and the home server approach described in the previous section is employed, then each server hosts all of the modules except the user interface manager that resides on the client. If, however, each client is expected to contact individual servers on its own, then, most likely, the clients will host the full user processor functionality while the data processor functionality resides in the servers.

1.7.10 Multidatabase System Architecture multidatabase systems (MDBS) represent the case where individual DBMSs (whether distributed or not) are fully autonomous and have no concept of cooperation; they may not even “know” of each other’s existence or how to talk to each other. Our focus is, naturally, on distributed MDBSs, which is what the term will refer to in the remainder.

Relational software uses the concept of database normalization and primary and foreign key constraints to establish relationships between rows of data in different database tables. Primary keys are unique identifiers for rows in a table; for example, a customer identification number could be the primary key in a table with data about a company's customers. Foreign keys point to primary keys in other tables.

In most current literature, one finds the term data integration system used instead. We avoid using that term since data integration systems consider non-database data sources as well.

The differences in the level of autonomy between the distributed multi-DBMSs and distributed DBMSs are also reflected in their architectural models. The fundamental difference relates to the definition of the global conceptual schema. In the case of logically integrated distributed DBMSs, the global conceptual schema defines the conceptual view of the entire database, while in the case of distributed multi-DBMSs, it represents only the collection of some of the local databases that each local DBMS wants to share.

In a MDBS, the GCS (which is also called a mediated schema) is defined by integrating either the external schemas of local autonomous databases or (possibly parts of their) local conceptual schemas. Furthermore, users of a local DBMS define their own views on the local database and do not need to change their applications if they do not want to access data from another database

The component-based architectural model of a multi-DBMS is significantly different from a distributed DBMS. The fundamental difference is the existence of full-fledged DBMSs, each of which manages a different database. The MDBS provides a layer of software that runs on top of these individual DBMSs and provides users with the facilities of accessing various databases.

Referencias:

Pech May, F., 2012. *Bases de Datos Distribuidas -Panorama General*. [online] Tamps.cinvestav.mx. Disponible en: <https://www.tamps.cinvestav.mx/~fpech/BDD/recursos/RecBDD/PreAnterior.pdf> [Consultado 8 marzo 2022].

Aisii.azc.uam.mx. n.d. *Arquitectura ANSI/SPARC - del AISII*. [online] Disponible en: <http://aisii.azc.uam.mx/areyes/archivos/Licenciatura/BDD/Arquitecturas.pdf> [Consultado 8 marzo 2022].

kryptonsolid, «¿Cuál es la diferencia entre DBMS y RDBMS?,» kryptonsolid.com, [En línea]. Available: https://kryptonsolid.com/cual-es-la-diferencia-entre-dbms-y-rdbms/. [Último acceso: 07 03 2022].

Tuxtepec, I. T. (2022, 9 de marzo). *BAses de datos heterogeneas*. slideshare. https://es.slideshare.net/juan_anaya/bdd-heterogneas