International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-1 Issue-2, December 2011

Design Distributed Database Strategies for SQMD Architecture

Shailesh R. Thakare, C.A. Dhawale, Ajay B.Gadicha

Abstract*: Database is not static but rapidly grows in size. These issues include how to allocate data, communication of the system, the coordination among the individual system, distributed transition control and query processing, concurrency control over distributed relation, design of global user interface, design of component system in different physical location, integration of existing database system security. The system architecture makes use of software portioning of the database based on data clustering, SQMD (Single Query Multiple Database) architecture, a web services interface and virtualization software technologies. The system allows uniform access to concurrently distributed database, using SQMD architecture. In this Paper explain Design Strategies of Distributed Database for SQMD architecture.

***Index Terms*: SQMD, Global User Interface**

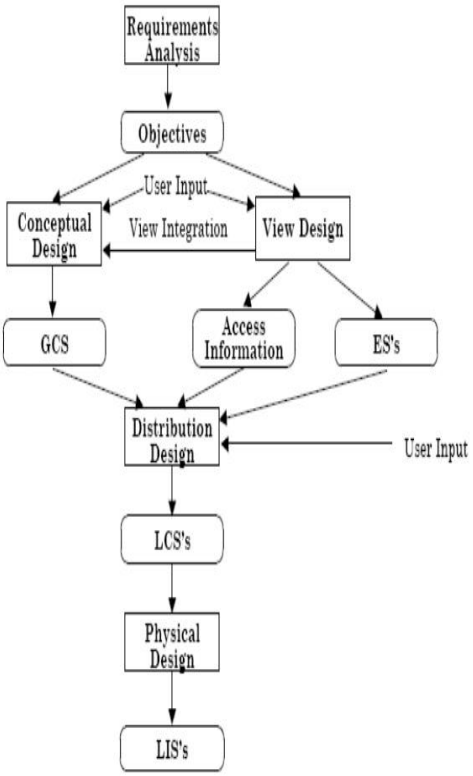
I. INTRODUCTION

Top-down and bottom up approach are the two major design strategies for distributed database design. Although these two approaches carry out very different design process, the necessity of applying one approach to complement another is possible since real applications are likely to be too complicated to fit in just one approach. The problems of effectively partitioning a huge dataset and of efficiently alleviating too much computing for the processing of the partitioned data have been critical factor for scalability and performance. In today's data deluge the problems are becoming common and will become more common in near future. The principle "Make common case fast" (or "Amdahl's law" which is the quantification of the principle) can be applied to make the common case faster since the impact on making the common case faster may be higher, while the principle generally applies for the design of computer architecture.

Our scalable, distributed database system architecture is composed of three tiers- a web service client (front-end), a web service and message service system (middleware), and finally agents and a collection of databases (back-end). To achieve scalability and maintain high performance, we have developed a distributed database system on virtual Private servers. The databases are distributed over multiple virtual private servers by fragmenting the data using two different methods: data clustering and horizontal partitioning to increase the molecule shape similarity and to decrease the query processing time. The distributed nature of the databases is transparent to end-users and thus the end-users are unaware of data fragmentation and distribution. The middleware hides the details about the data distribution. To support efficient queries, we used a single Query Multiple Database (SQMD) mechanism which transmits a single query that simultaneously operates on multiple databases, using a publish/subscribe paradigm. A single query request from end-user is disseminated to all the databases via allow high performance interaction between users and huge database by building a scalable, distributed database system using virtualization technology.

II. DESIGN STRATEGIES & DISTRIBUTED DATABASE ARCHITECTURE

Figure-1: Top-down Design Process



Manuscript published on 30 December 2011.
***** Correspondence Author (s)
Prof. Shailesh R. Thakare*, Professor, Department of Computer Science & Engineering, P.R.Pote(Patil) College of Engg & Mgmt, Amravati (Maharastra), India (e-mail: shaileshthakar2003@yahoo.com).
Dr. C.A. Dhawale, Professor, Department of Computer Science & Engineering, P.R.Pote(Patil) College of Engg & Mgmt, Amravati (Maharastra), India (e-mail: cadhawale@rediffmail.com).
Ajay B. Gadicha, Asst. Professor, Department of Computer Science & Engineering, P.R.Pote(Patil) College of Engg & Mgmt, Amravati (Maharastra), India (e-mail: ajjugadicha@gmail.com).

A. *Top Down Approach*

Top-down design process is mostly used in designing system from scratch. Figure illustrates the process of top-down design. The process starts from a requirement analysis phase including analyzing of the company situation, defining problems and constraints, defining objectives, and designing scope and boundaries. The next two activities are conceptual design and view design. Focus on the data requirements, the conceptual design deals with entity relationship modeling and normalization. It creates the abstract data structure to represent the real world items. The view design defines the user interfaces. The conceptual schema is a virtual view of all databases taken together in a distributed database environment. It should cover the entity and relationship requirement for all user views. Furthermore, the conceptual model should support existing applications as well as future applications. The definition of the global conceptual schema (GCS) comes from the conceptual design. The next step is distribution design. The global conceptual schema and the access information collected from the view design activity are inputs of this step. By fragmenting and distributing entities over the system, this step designs the local conceptual schemas. Therefore, this step can be further divided into two steps: fragmentation and allocation. Distribution design also includes the selection of DBMS software in each site. The mapping of the local conceptual schemas to the physical storage devices is accomplished through the physical design activity. Throughout the design and development of the distributed database system, constant monitoring and periodic adjustment and tuning are also critical activities in order to achieve successful database implementation and suitable user interfaces.

B. *Bottom up Approach*

Bottom-up approach is suitable when the objective of the design is to integrate existing database systems. The bottom-up design starts from the individual local conceptual schemas and the objective of the process is integrating local schemas into the global conceptual schema. One of the most important aspects of design strategy is to determine how to integrate multiple database system together. Implementation alternatives are classified according to the autonomy, distribution, and heterogeneity of the local systems.

Autonomy indicates the independency of individual DBMS. In the autonomous system, the individual DBMS are able to perform local operations independently and have no reliance on centralized service or control. The consistency of the whole system should not be affected by the behavior of the individual DBMS. Three possible degrees of autonomy are tight integration, semiautonomous system, and total isolation. In a system which is tightly integrated, although information is stored in multiple databases, users only see a single image of the entire system. One of the DBMS controls the processing of the user request. The DBMS in semiautonomous system can operate separately and they are also willing to share their local data. In total isolated systems, individual DBMS do not know the existence of other DBMS.

The physical distribution of data over multiple sites is another characteristic of distributed databases. Distributed system can be classified as client/server distribution or peer-to-distribution based on how the data are distributed and how to manage them. Heterogeneous DDBMS integrate multiple independent databases into a single distributed database system and provide transparency of the heterogeneity. Individual DBMS can implement different data model, use different query language and transaction management protocols.

Moving along the distribution dimension, the client/server distribution is introduced when the system is distributed with an integrated view providing to users. This implementation requires assigning the control of the entire system to one DBMS (single server) or several DBMS (multiple servers). The server(s) control each user request although the request might be serviced by more than one DBMS. The other scenario is that the system is fully distributed, but the distribution is transparent to the user. Each DBMS provides the identical functionality and there is no distinction among clients and servers. In distributed system, external schemas are defined as being above a global conceptual schema (GCS) which describe the logical data structure of the entire system. The global conceptual schema provides distribution transparency to users. It is the union of local conceptual schemas of local database systems which describe the logical organization of data at each site. The physical data organization on each site in the system is presented by local internal schema.

III. ARCHITECTURE FOR SINGLE QUERY MULTIPLE DATABASE (SQMD) MECHANISM

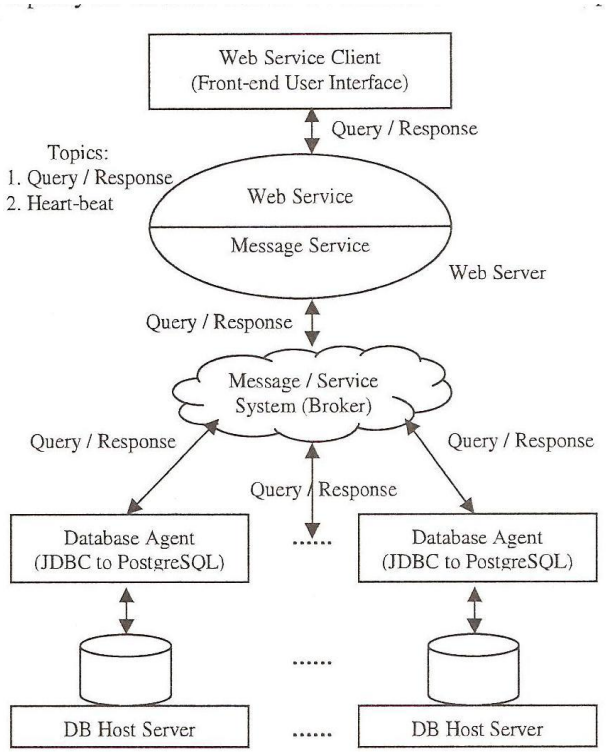


Figure-2: shows a broad 3-tier architecture view for our scalable distributed database system

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

The scalable, distributed database system architecture is composed of three tiers-the web service client (front-end), a web service and message service system (middleware), agents and a collection of database (back-end). The virtualization environments based on open VZ and software (with web service SQMD using publish/subscribe mechanism, and data clustering program) architecture concentrate on increasing scalability with increased size of distributed data, providing high performance service with the enhancement of query/response interaction time, and improving data locality. Our message and service system,, which represents a middleware component, provides a mechanism for simultaneously.

A. *Web Service*

A web service is a software platform to build applications running on a variety of platforms as services. The communication interface for web service is described by XML that follows SOAP (simple Object Access Protocol) standard. Other heterogeneous systems can interact with the web service through a set of descriptive operations using SOAP. Web services we used the open-source Apache Axis library which is an implementation of the SOAP specification. Also we used WSDL (web service Description Language) to describe our web service operations. A web service client reads WSDL to determine what functions are available for database service one service is query request service and the other is reliable service which detects whether database servers fail.

B. *Web Service Client (Front-End)*

Web service clients can simultaneously access (or query) the data in several database in a distributed environment Query requests from clients are transmitted to the web service, disseminated through the message and service system to database servers via database agents. The well known benefits of the three-tier architecture results in the web service clients do not need to know about the individual distributed database servers, but rather, send query request to a single web service that handles query transfer and response.

C. *Message and Service Middleware System*

For communication service between the web service and middleware, and the middleware and database agents, we have used Narada Brokering for message and service middleware system as overlay built over heterogeneous networks to support group communications among heterogeneous communities and collaborative applications. The Narada Broking from Community Grids Lab (CGL) is adapted as a general event brokering middleware, which supports publish/subscribe messaging model with a dynamic collection of brokers and provides services for TCP, UDP, Multicast, SSL, and raw RTP clients. The Narada Brokering also provides the capability of the communication through firewalls and proxies. It is an open source and can operate either in a client-server mode like JMS or in a completely distributed peer-to-peer-mode. This paper we use the terms “message and service middleware (or system)” and “broker” interchangeably. Database Agent

In Figure 2, the database agent (DBA) is used as a proxy for database server (Postgre SQL). The DBA accepts query requests from front-end users via middleware, translates the requests to be understood by database server sand retrieves the front-end- user via message service system (broker) and web service. Web service clients interact with the DBA via middleware, and then the agent communicates with Postgre SQL database server. The agent has responsibility for getting responses from the database server and performs any necessary concatenations of responses occurred from database for the back-end (database server), the agent retains communication interfaces (publish/subscribe) and thus can offload some computational needs.

D. *Database Server*

A number of data partitions split by data clustering or horizontal partitioning method are distributed into Postgre SQL database servers. Another benefit of database servers based on the three-tier architecture is that they do not concern about a large number of heterogeneous web service clients but need to only process queries and return results of the queries. We used the open source database management system Postgre SQL. With such an approach using open management system, we can build a sustainable high functionality system taking advantage of the latest system, we can build a sustainable high functionality system taking advantage of the latest system technologies with appropriate interface between layers (agents and database host servers) in a modular fashion.

IV. DISTRIBUTED DESIGN ISSUES

The design of a distributed database introduces the following interrelated issues. These issues complicate distributed database design.

- How to partition the database into fragments
- How many copies of a fragment should be repliced
- How to allocate the fragments and replicas

A. *Fragmentation*

Data fragmentation allows us to fragment relations to appropriate units of distribution since usually applications are only deal with a subset of a relation. Each fragment can be stored at any site across the network. The decomposition of a relation enables the concurrent execution of several transactions. Data fragmentation information is stored in the distributed data catalog for accessing by the transaction processor to process user requests. There are three types of fragmentation strategy: Horizontal, Vertical, and Mixed fragmentation.

Horizontal fragmentation divides a table into subsets of tuples (rows) based on the database information and application information. Each fragment consists of unique rows and is stored at a different site. The advantage of horizontal fragmentation is that it allows data locality by storing the fragments in the sites where they are most frequently accessed. Parallel processing is allowed on fragments of a relation.

An example of horizontal fragmentation can be that a bank fragments its customer table by location.

Fragment. For example, a customer table can be divided into two fragments with one has customer ID, name,

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

address and another has customer ID and other financial information.

Vertical fragmentation can be achieved using two strategies: grouping and splitting. The former creates one fragment for each attribute and groups them to construct fragments. While the latter uses a top-down approach which progressively splits global relationship into fragments.

Vertical fragmentation is more complex than horizontal fragmentation and has been used in both centralized and distributed environments. Vertical fragmentation is the equivalent of PROJECT statement. It divides a table into attribute (column) subsets. Therefore it need define subsets of attributes that are commonly jointly accessed in order to support efficient accessing and transferring data. Each fragment is located at a different site and unique attributes except that the key column has to be in every mixed fragmentation is the combination of horizontal and vertical strategies. A table can be divided into subsets of rows, each having a subset of columns. Using the same example, a bank can partition its customer table by location and grouped by certain attributes.

All the fragmentation algorithms must satisfy three correctness criteria:

- Completeness: Fragmentation of a relation is complete if and only if each data item in the relation appears in at least one fragment. This criterion ensures that there is no data loss during fragmentation.
- Reconstruction: It must be possible to use a relational operation to reconstruct the original relation to ensure the preservation of functional dependencies.
- Disjointness : Each data item is found in only one fragment to minimize redundancy. Vertical fragmentation is the exception to this rule in that primary key columns must present in every fragment to allow reconstruction.

B. Replication

In a distributed database, a relation or a fragment can be replicated or copied. Copies of data may be stored redundantly in two or more sites to serve specific information requirements and enhance data availability. For example, a bank can have copies of a customer table fragment stored in the database of its branch and also in the headquarter database.

Data replication decision should consider the size of database and data usage frequency. A fully replicated database stores multiple copies of each fragment at multiple sites. While in a partially replicated database, only some fragments are replicated. A database is fully redundant if each site contains a copy of the entire database.

Replication has several advantages. It improves the data availability and response time. It reduces the cost of accessing and transferring data. Transactions can be executed parallel using replicas of relation or fragment. However, data replication increases the cost of updates since all replicas have to be updated to ensure they are all identical. Therefore, replication increases the complexity of the concurrency control.

C. Allocation

Data allocation involves the processing of determining the location of the fragments based on the information of the database, the types of transactions to be applied to the database, the communication network, the storage capability of each site, and the design goal of cost, response time and data availability.

V. CONCLUSION

Architecture of SQMD performs parallel operations on the multiple databases using single query which explores the clear intension to designing this strategy, which strategy to decide being convenient as per the requirement.

REFERENCES

1. Kangseak Kin, Rajarshi Guha,Marton E.Pierce,Geoffrey C.Fox,Divid J.Wild Kevin E.Gilbert"SQMD:Architecture for Scalable, Distributed Database System built on Virtual Private Ser vers",{Kakim rguha, Marpire, gef, djwild, gilben}@indiana.edu
2. Dixu "Distributed Database System Design" Minnesota State University Kankato.
3. Ahmet Uyar, Wenjun Wu, Hasan Bulut, Geoffrey Fox. Service-Oriented Architecture for Building a Scalable
4. Videoconferencing System March 25 2006 to appear in book "Service-Oriented Architecture - Concepts & Cases"
5. published by Institute of Chartered Financial Analysts of India (ICFAI) University.
6. Apache Axis2, http://ws.apache.org/axis2/
7. Ballester, P.J., Graham-Richards, W., J. Comp. Chem., 2007, 28, 1711-1723.
8. Cassandra project, http://code.google.com/p/the-cassandra-project/
9. Chaitanya K. Baru, Gilles Fecteau, Ambuj Goyal, Hui-I Hsiao, Anant Jhingran, Sriram Padmanabhan, George P. Copeland,
10. Walter G. Wilson. DB2 Parallel Edition. IBM System Journal, Volume 34, pp 292-322, 1995.
11. Chembiogrid (Chemical Informatics and Cyberinfrastructure Collaboratory),
12. http://www.chembiogrid.org/wiki/index.php/Main_Page
13. Ciaccia, P., Patella, M., Zezula, P., Proc. 23rd Intl. Conf. VLDB, 1997.
14. Community Grids Lab (CGL), http://communitygrids.iu.edu
15. Dalby, A., Nourse, J., Hounshell, W., Gushurst, A., Grier, D., Leland, B., Laufer, J., J. Chem. Inf. Comput. Sci., 1992, 32,