Volume 4, Issue 10, October 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Implementation of Distribution Transparency in Heterogeneous Distributed Database System Using Aglet

Prof. C. M. Jadhav*
CSE & Solapur University
Maharashtra, India

Bhaskar R Nadargi
CES & Solapur University
Maharashtra, India

Abstract: ***The definition of DDBMS states that the system should make the distribution transparent to the end user and application programmer. Purpose of transparency in DDBMS is to hide implementation details of distributed database system. The main objective of implementing distribution transparency in heterogeneous distributed database system is to offer transparent access to data no matter where in the network the data is located by providing single system image to end user. Heterogeneity in distributed database system refers to system containing multiple databases of different kinds. It is necessary to implement distribution transparency because applications are written depending on how much distribution transparency is provided by the DDBMS.***

To provide transparency in DDBMS several solutions are proposed like Use of Name Server (Centralized Scheme) and Use of Aliases. But these methods have some limitations which can be efficiently overcome by using new emerging technology called as AGLET. AGLETS are mobile agents which are capable to migrate from one system to another system of the network carrying code and data with it. Here we propose methodology that makes use of this property of AGLET to provide distribution transparency in heterogeneous distributed database system.

Keywords— Distributed database, Mobile agent, Aglet, Distribution Transparency, Heterogeneous database

I. INTRODUCTION

Presently network architectures are becoming more and more complicated due to heterogeneity of the network components. In our work heterogeneity refers to the different database vendors. In this scenario we mainly focus on different levels of distribution transparencies present in heterogeneous distributed database systems and their implementation with the help of methodology mentioned in Methodology section. Transparency in DDBMS allows end user to access distributed database system same as centralized database system. In other words, users are made to believe that they are communicating with centralized DBMS; all complexities of a distributed database system are kept transparent to the user.

Following terminologies are important to understand before going for implementation [10].

1.1 Distribution Transparency:

Distribution transparency comes in picture when data is distributed among multiple databases i.e. when distributed database system is used for storing the data. If a DDBMS exhibits distribution transparency then user does not need to know about:

a) Presence of replication of data. That is need not to do any explicit activity to keep consistency among all data copies
b) Location of data or data fragment
c) Partitioning technique used to distribute data among multiple distributed databases.

The level of transparency supported by the DDBMS varies from system to system. Following are recognized levels of distribution transparency:

• **Fragmentation transparency:**

It is the highest level of distribution transparency. Application programmer does not need to know about how database is partitioned before accessing it. Therefore, neither fragment locations nor fragment names are specified in query. If transparency up to fragmentation transparency level is implemented in DDBMS then all lower level transparencies are automatically gets implemented.

• **Location transparency:**

When application programmer specifies the database fragment name in query but does not specify fragment's location then this type of transparency is called as distribution transparency up to location transparency.

• **Local mapping transparency:**

It exists when application programmer specifies both the fragment names and their locations in query.

Transparency features can be summarized as below.

Table1: Summarized Transparency Features

Fragment Name	Location Name	DBMS Supports Transparency	Level Of Transparency
Yes	Yes	Local Mapping	Low
Yes	No	Location transparency	Medium
No	No	Fragmentation Transparency	High

There is no reference in above table to a situation in which the fragment name is "No" and the location name is "Yes." The reason for not including that scenario is simple: you cannot have a location name that fails to reference an existing fragment.

1.2 Why to Use Aglets?

All above defined levels of distribution transparencies can be efficiently implemented by making use of Aglet discovered by IBM. Aglet is java based mobile agent that travels to aglet-enabled host in a computer network. It is autonomous, since it runs in its own thread of execution after arriving at a host and reactive because of its ability to respond to incoming messages [6].

Main Advantages of Using Aglets:

1. It eliminates drawbacks of Centralized Name Server Scheme which are potential performance bottleneck and single point of failure of system.
2. Better solution over use of aliases scheme which fails to achieve network transparency.
3. Reduce network traffic
4. Overcome network latency
5. Make better use of resources
6. Execute asynchronously and autonomously
7. Adapt dynamically
8. Robust and fault tolerant

II. RELATED WORK

AMIT P. SHETH and JAMES A. LARSON [1] presented paper on Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. They have provided reference architecture for distributed database management systems from system and showed how various FDBS architectures can be developed. They also discussed critical issues related to developing and operating an FDBS.

Tuomas Sandholm and Qianbo Huai [11] presented paper on Nomad: Mobile Agent System for an Internet-Based Auction House. In this they brilliantly used mobile agents for Internet auction called as eAuctionHouse. They explained how mobile agent can travel to eAuctionHouse and participate in auction on behalf of user.

Danny B. Lange and M. Oshima [6] presented paper on Mobile Objects and Mobile Agents in World Wide Web Journal, 1998. In this they explained why java is one of the powerful tool for developing mobile agents. Here they studied java based mobile agent called as 'Aglet' and explained detailed characteristics of aglet. Finally they proposed design model of aglet which can be used in many application as mobile agent.

A Di Stefano, L Lo Bello, C Santoro [3] Locating Mobile Agents in a Wide Distributed Environment proposes a naming scheme and a location protocol for validating and locating mobile agents in distributed environment. As finding the position of a mobile agent in a wide distributed system still represents an open research issue they studied existing mobile agent platforms implement ad hoc naming and location policies and proposed suitable agent naming scheme and location finding protocol.

A Di Stefano, L Lo Bello, C Santoro [2] have given an idea about a distributed heterogeneous database system based on mobile agents. They proposed model of distributed transactions as a set of mobile agents and present relevant execution semantics. They also guaranteed to follow ACID properties with this model.

YIN-FU HUANG AND JYH-HER CHEN [12] worked on the concept of Fragment Allocation in Distributed Database Design. They have given some issues on allocation of fragments like How a global relation should be fragmented, How many copies of a fragment should be replicated, How fragments should be allocated to the sites of the communication network, What the necessary information for fragmentation and allocation is.

III. PROPOSED WORK

Our aim is to provide high level of distribution transparency as described in Table1 i.e. to provide transparency up to fragmentation level which indirectly covers all levels of distribution transparencies.

We are achieving this by considering horizontally partitioned database, disallowing replication of data fragments maintaining single copy of data throughout the system and distributed database system consists of three databases of different kinds. That is our overall DDBMS is Heterogeneous Distributed Database System with horizontal partitioning without replication of fragments. All fragments of each table would have same name and no two fragments of same table are stored on same database server i.e each database server can have at most one fragment of every table. Our architecture will look like as below containing some components performing special tasks.

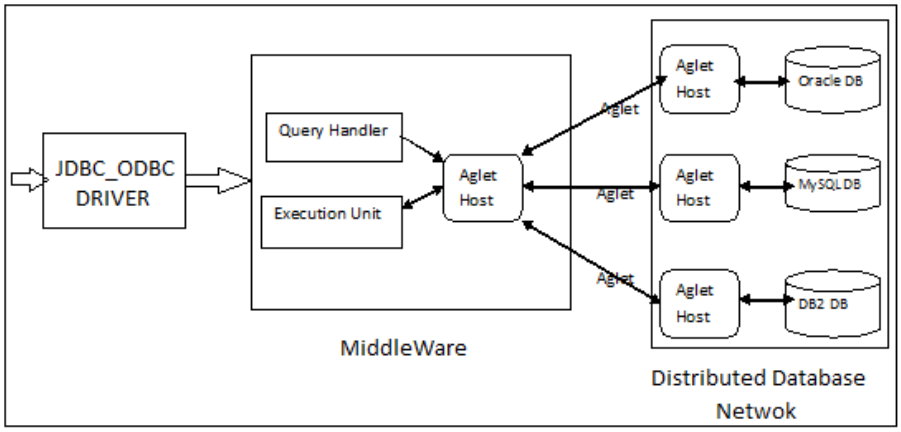

Figure 1: Architecture of Heterogeneous distributed database system.

Middleware:
It is an intermediate system between application and distributed database servers. It plays an important role in hiding distributed database system image from end user. It consists of few components within it performing some special tasks.
JDBC-ODBC Driver:
We maintain connection to all database systems using only one logical connection string through JDBC-ODBC driver. So there is no need to maintain separate connections to each database by application.
Query Handler:
It is responsible for translating query over global schema to local schema. It accepts the single query over global schema and translates it to multiple queries over local schema.
Execution Unit:
It performs condition checks over query and generates the final result i.e. obtains set of tuples from result set satisfying mentioned condition in query.
Aglet Hosts:
It generates aglets and handles transmission of aglets over the network. Aglet hosts on remote system accepts incoming aglets and resumes its execution on system. It is mandatory to have aglet host on each database servers.
Database network:
Network connecting all distributed Heterogeneous databases together forms database network. Connection between databases is required to allow aglet to move from one database to other databases if required.

IV. METHODOLOGY

Application programmer writes the query considering there is only one single database in network. We hide our distributed database system from programmer by providing single logical connection string to connect database network. We maintain connection between application and database network through middleware by overriding the methods of JDBC_ODBC connection class in Java. Query defined in application is over the global schema and is referred as "Global Query". This single global query is transformed to multiple separate queries at middleware as per distribution of data done by database partitioning scheme which is transparent to user. This process is called as "Transformation of Global queries to Fragment queries". Then these fragment queries are sent to individual database using aglets for execution. Aglet host running on each database server is responsible for executing queries on receipt of aglet and sending the result of query back to middleware using aglet. Finally result is sent to user machine (client) as output of global query.
Depending on type of application we consider following two cases [10]:

A) Distribution Transparency for Read-Only Applications:
Read-only applications are the applications which are generally used to generate reports i.e. involves only reading of database as per criteria or requirement. These applications contain only SELECT queries. Consider following tables with their schemas:

ORDERS (ORDER_ID, CUST_ID, DATE);
CUSTOMERS (CUST_ID, NAME, ADDRESS, CONTACT_NO, TYPE, STATUS);
As per our architecture above tables are horizontally partitioned and their fragments with same name are stored among three databases.
Now consider select query to find all order details for all customers as below.

SELECT CUSTOMERS.NAME, CUSTOMERS.CONTACT_NO, ORDERS.ORDER_ID, ORDERS.DATE FROM CUSTOMERS, ORDERS WHERE CUSTOMERS.CUST_ID=ORDERS.CUST_ID;
This SELECT query is received by middleware via JDBC-ODBC connection string and forwarded to query handler. Here query is processed in two steps. In first step depending on number tables present in where condition initial queries are generated as below:

SELECT CUSTOMERS. CUST_ID FROM CUSTOMERS;
SELECT ORDERS.CUST_ID FROM ORDERS;

Now we store these intermediate queries into aglet and send it to each database server using aglet host. Aglet host present at database server receives this aglet and executes queries stored in aglet. Results of these queries are stored into aglet and these aglets are sent back to middleware by all database servers for further processing. After receiving required data from all databases, execution unit applies condition mentioned in WHERE clause i.e. **(CUSTOMERS.CUST_ID= ORDERS.CUST_ID)** on received data. Finally we get list of common CUST_IDs present in both tables as per condition.Now in second step we generate final queries which retrieve final data required by original select query written in application as below.

SELECT CUSTOMERS.NAME, CUSTOMERS.CONTACT_NO FROM CUSTOMERS
WHERE CUSTOMERS.CUST_ID IN (, ,);
SELECT ORDERS.ORDER_ID, ORDERS.DATE FROM ORDERS WHERE ORDERS.CUST_ID IN (, ,);

Here 'In' clause contains all CUST_IDs obtained in first step. Then procedure mentioned in first step is repeated in order to execute above queries. Finally results of both queries are merged and send it to application. In this way select queries can be processed without revealing presence of distributed database.

B) Distribution Transparency for Update Applications:

This type of applications perform insert, delete and update operation on records of database. It involves INSERT, DELETE and UPDATE queries.

Processing of INSERT and DELET queries are quite simple than update queries. We create separate queries for each table mentioned in actual INSERT or INSERT query and send them to each database server using aglet. At database server INSERT and DELETE queries are executed as usual. While executing INSERT query data will be inserted into fragment if it satisfies portioning criteria. Thus INSERT query will be executed successfully at only one server where it satisfies partitioning criteria and avoids duplicate insertion of records.

Implementing distribution transparency for update application containing update queries is more complex task than read-only applications. This is because we need to take care of shifting tuple to new fragment after updating attribute value on which partitioning is done and keeping all copies of data consistent in case if replication of data is supported. As we are not supporting replication of data so second scenario is not applicable in our case. Consider update query as below.

UPDATE CUSTOMERS, ORDERS
SET CUSTOMERS.STATUS='NA', ORDERS.DATE=31.12.2014
WHERE CUSTOMERS.CUST_ID=ORDERS.ORDER_ID AND CUSTOMERS.TYPE='A';

Above Update query is processed in same way as select query is processed. In first step following intermediate queries are generated depending on number of tables and fields used in condition and result is obtained for the same.

SELECT CUSTOMERS. CUST_ID, CUSTOMERS.TYPE FROM CUSTOMERS;
SELECT ORDERS.ORDER_ID FROM ORDERS;

Execution unit applies condition **(CUSTOMERS. CUST_ID=ORDERS.ORDER_ID AND CUSTOMERS. TYPE='A')** on received data and obtains set of entries satisfying above condition. In second step following queries are generated for final updation as expected by original update statement written in application.

UPDATE CUSTOMERS SET CUSTOMERS.STATUS='NA' WHERE CUSTOMERS.CUST_ID IN (, ,) AND CUSTOMERS.TYPE IN (, ,);
UPDATE ORDERS SET ORDERS. DATE = 31.12.2014 WHERE ORDERS.ORDER_ID IN (, ,);

Set of entries satisfying condition is used as input for IN clause in above queries. These queries are stored in aglet and sent to each database for performing final updation.

In this way without mentioning fragment's name and location, queries can be written. Thus we succeeded in hiding actual image of distributed database system from end user and eventually supporting all levels of distribution transparencies efficiently for both read only applications and update applications.

If we do not support distribution transparency then programmer have to write queries in application as below.

INSERT INTO ORDERS3 VALUES (1023, 110, 22.01.2014);
DELETE FROM CUSTOMERS2 WHERE CUSTOMERS2.NAME='KAT';

Where ORDERS3 and CUSTOMERS2 are the fragment's names of table's ORDERS, CUSTOMERS stored at database servers 3 and 2 respectively. In such cases programmer must have information about names and locations of all fragments present in database system which reveals presence of distributed database system and fails to achieve primary purpose of distributed database system.But if we use aglet technology in implementing distribution transparency then it is not necessary to maintain information about fragment's location or fragment's name. We create aglet for each query and send it to all database servers irrespective of location of their fragments after processing them at middleware server. Receiving aglets and processing of query stored inside the aglet is handled by Aglet Host of respective servers. Thus task of maintaining fragment's name and location information and writing queries according to them is totally eliminated and making implementation simple efficient and scalable.

V. CONCLUSIONS

Some of the current DDBMS implementations impose limitations on the level of transparency support. For instance, you might be able to distribute tables but not the data across multiple sites appropriately. Which means DDBMS supports location transparency but not fragmentation transparency. But introducing Aglet technology in DDBMS we can overcome limitations of currently available technologies like Centralized Scheme-Name Server and Use of Aliases. Thus use of aglets for implementing distribution transparency in distributed heterogeneous database system can be the unique and reliable solution because of its numerous advantages over existing solutions. Here we have totally eliminated the use of mobile agents locating protocol and mobile agent naming concept required for agent validation. These both things are supported by Java by implementing Aglet-API [6]. Moreover in future with some minor changes we can support addition and deletion of one or more databases from distributed system. For future enhancement we can introduce query optimization techniques and can provide good response time for more complex queries with the help of aglets.

REFERENCES

[1] AMIT P. SHETH and JAMES A. LARSON – "Federated database systems for managing distributed, heterogeneous, and autonomous databases', ACM Computing Surveys, Volume 22 Issue 3, Sept. 1990.

[2] Antonella Di Stefano, Lucia Lo Bello, Corrado Santoro – 'A Distributed Heterogeneous Database System based on Mobile Agent", Proceedings of the 7th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, p.223-229, June 17-19, 1998

[3] Antonella Di Stefano, Corrado Santoro – "Locating Mobile Agents in a Wide Distributed Environment", IEEE Transactions on Parallel and Distributed Systems, Volume 13 Issue 8, August 2002

[4] Aridor, Y. and M. Oshima – "Infrastructure for Mobile Agents: Requirements and Design", in Lecture Notes in Computer Science (1477), Springer, 38-49, 1998

[5] Baumann J, Hohl F, Rothermel K, Straber M. Mole – **"**Concepts of a mobile agent system". World Wide Web 1998; 1(3):123–137. .

[6] D.B. Lange and M. Oshima – "Mobile agents with Java: The Aglet API," World Wide Web Journal, 1998.

[7] M.Madhavaram, D. L Ali and Ming Zhou – **"**Integrating heterogeneous distributed database systems. Computers & Industrial Engineering", 31(1– 2):315–318, October 1996.

[8] Ozsu and Valduriez, M. T. Ozsu and P.Valduriez – "Principles of Distributed Database Systems", Prentice- Hall, Englewood Cliffs, NJ, 1991.

[9] S.Papastavrou, G. Samaras and E. Pitoura. Mobile Agents for WWW Distributed Database Access. In Proceedings of the Fifteenth International Conference on Data Engineering, March 1999.

[10] Stefano Ceri and Giuseppe Pelagatti – **"**Distributed Database Principles & Systems" Reference book.

[11] Tuomas Sandholm , Qianbo Huai – **"**Nomad : Mobile Agent System for an Internet-Based Auction House", IEEE Internet Computing, v.4 n.2, p.80-86, March 2000

[12] Yin-Fu Huang, Jyh-Her Chen, Fragment Allocation in Distributed Database Design. Journal of Information Science and Engineering. 2001, 17 P. 491-506

