Jan Janech – Tomas Baca *

DISTRIBUTED DATABASE SYSTEMS IN THE DYNAMIC NETWORKS ENVIRONMENT

This article describes a new architecture of a distributed database system, which is designed especially for the dynamic networks environment. Problems with the architecture of a traditional distributed database system are explained, and a solution is offered. The architecture introduced in this article does not intend to be a universal database system. It is a specialized solution for the implementation of distributed database systems in the dynamic networks environment.

1. Introduction

During 1980s distributed database systems (DDBS) went through turbulent evolution. Many various architectures and principles were developed most of them not very successful. Evolution slowed down in early 1990s [1, 7, 8]. In 1991 Valduriez and Ozsu published their book Principles of Distributed Database Systems [7] now considered a "bible" of DDBS.

Since then almost nothing has changed in the architecture of DDBS. Research has been focused on transaction management, replication management and fragmentation of database and basic principles remained untouched. However demands have changed. Many things, thirty years ago almost impossible, are now considered normal. Network architecture has changed; wireless networks are used much more often which brings complications like lower communication speed, more vulnerable channel and higher probability of interruption, etc.

These problems are much more significant in ad-hoc networks. Nodes in ad-hoc network are interconnected without any plan or topology. From the network's point of view any node can be connected and it will always work with the same basic principles. Traditional architectures of DDBS are not capable of working in such environment because complete detailed knowledge of the whole system and location of data is required.

We have already proposed a solution for these problems in [4]. In this paper we will go with the topic further and focus on processing of queries in the proposed system. However the basic principles of the new architecture are explained in the next section.

2. Basic Principles of the New Architecture

Traditional DDBS make use of Global Directory/Dictionary (GD/D), which stores mapping between local and global conceptual schemas of every site. Without the mapping the system would not know where the data are located and how to query them. Although improving performance in wired networks, GD/D poses a great barrier for DDBS being deployed in dynamic wireless network. Moving nodes bring frequent changes of network topology, thus the GD/D is difficult to maintain up-to-date. We have therefore proposed to replace GD/D by a different principle.

In a dynamic network every site knows its immediate surroundings only. So querying of a distributed database is fairly limited in such environment. The only sites which can be addressed to with queries, except the one which requests information, are those in the immediate surroundings in the network. This way the system naturally creates virtual clusters of sites that can communicate with each other. The clusters tend to overlap, thus one site usually can communicate with different set of sites than another site belonging to the same cluster.

This implies a possibility to move the directory from global level to cluster level. We call this principle Cluster Directory/Dictionary (CD/D). The directory contains only mapping of the global conceptual schemas of those sites that are accessible to the local conceptual schemas exclusively. However, there might be problems in this system due to clusters overlapping. Therefore the local conceptual schema of each site has to be mapped onto the global conceptual schema in several CD/D. Still, the question remains how to store CD/D in the cluster because there is no central site in the

* **Jan Janech, Tomas Baca**
Department of Software Technologies, Faculty of Management Science and Informatics, University of Zilina, Slovakia,
E-mail: jan.janech@kst.uniza.sk

cluster, as it is strictly peer-to-peer system. The only possible way is to store it locally as described in [4]. Every site would have a mapping from the global conceptual schema onto its own local conceptual schema. The CD/D then consists of all of the local mappings in the cluster.

Executing local queries in such a system is very simple because all of the accessible data and all of the mappings are available, so there is no need to communicate with the other sites in the cluster. But since the queries are normally done globally across the whole accessible part of the database, the communication between the sites in the cluster is necessary.

3. Query Processing

Before a site can process a query, it would need complete CD/D, which is stored fragmented, every site in the cluster keeps its own local copy. Before the execution plan can be created the query site has to gather all the local copies. It would use broadcast messages. The whole process of querying is depicted by UML sequential diagram in Fig. 1. It can be separated to two sub-sequential phases:

1) *Gathering of CD/D.* Firstly the query site has to gather and process local fragments of CD/D. Because of dynamics of the network, it is possible that there would be no one to respond to the request or that the communication would be disrupted before it had been finished successfully. Therefore the query site has to stop waiting for the responses after a limited span of time. This phase consists of following steps:
   a) *Request for CD/D.* It is sent by query site in broadcast message. The query site also sets a period of time for waiting for responses.
   b) *The request is processed by other sites in the cluster.* Every data site receives the request and responses with its fragment of CD/D to query site in unicast message.
   c) *Processing of CD/D fragments.* After expiring the predefined period of time, the query node composes complete CD/D from received fragments.

2) *Executing the query.* With the knowledge of complete CD/D, the query node can start to execute the query. Firstly the query has to be decomposed to parts that can be executed by individual data sites. These components – subqueries are then sent to data sites. Subqueries are sent in unicast messages. After receiving responses the query node has to further process received data, because they are not necessarily what were originally requested for. Before returning them to the user, they have to be transformed into required form. This phase consists of following steps:
   a) *Global query optimisation.* This step can be executed even in the first phase. The most important is to minimise the amount of data that is to be sent back by data sites.

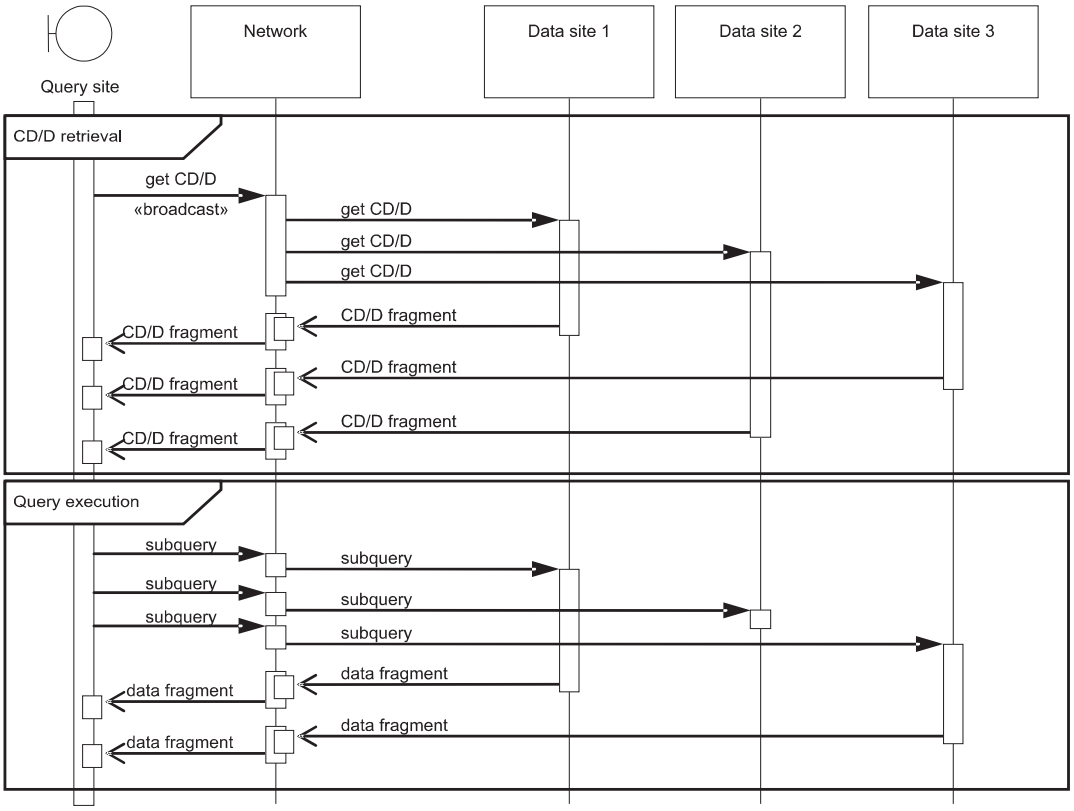

Fig. 1 Processing a query in the cluster

b) *Fragmentation of the query to subqueries.* The query site decomposes the query, set by the user, to subqueries addressed individually to data sites.
c) *Sending subqueries to data sites.* The query site sends subqueries to data sites in unicast messages. In respect of possible change of the cluster structure that might have occurred in the meanwhile, the query site has to set a span of time for waiting for responses.
d) *Local query optimisation.* Every data site optimises received subquery.
e) *Execution of subquery.* Every data site executes the subquery in its local part of the database that is accessible to it.
f) *Responding.* Every data site responds to the query site with requested data in unicast message.
g) *Evaluation of responses.* Received responses are not necessarily response to the request set by the user. The query site has to synthetise responses together.

This mechanism is rather complicated. The query site has to send two requests. The first one requesting CD/D is sent to all in one broadcast message. The second one requesting the data themselves is sent individually to every data site in unicast messages. The whole system has to be robust enough to deal with possible change of cluster structure at any time. Two types of changes can occur during the query is being processed:

- A new data site *joins the cluster* right after gathering of the CD/D. The query site would not however ask it for data because it does not know about its presence.
- A data site *leaves the cluster* right after gathering of the CD/D. The query site however counts with its presence while globally optimising the query and would have to go about evaluation of responses despite of missing response.

Therefore we simplified the process into the form depicted by UML sequence diagram displayed in Figure 2. The whole query is sent to the whole cluster in one broadcast message. Thus there is only one phase of processing the query, composed of the following steps:

a) *Global optimisation of the query.* The most important is to minimise the amount of data that is to be sent back by data sites.
b) *Sending the query.* The query node sends optimised query in a broadcast message. It will start to wait for responses for a limited span of time. The span of time must be estimated according to expected amount of data to be received. Every data site has received complete query.
c) *Fragmentation of the query.* Every data site decomposes the query and searches for subqueries, which it can locally execute.
d) *Local optimisation of subqueries.* Every data site locally optimises found subqueries.
e) *Execution of subqueries.* Every data site executes found subqueries.
f) *Responding.* Every data site responds with results of individual subqueries. However it also has to pack in executed subquery to every response.
g) *Evaluation of responses.* After the query site has finished waiting for responses, it synthetises final result of the query from received responses.

This algorithm does not suffer from aforementioned problems. The query site does not work with CD/D at all and needs only knowledge of Global Conceptual Schema. In case of cluster structure change during processing of a query receives the query site different set of data than it would receive if the cluster stayed unchanged. However the query site does not know about the change and therefore does not have to deal with it.

Each of both approaches would give the same results. The difference is that the former one is more complicated and includes more exceptional states it has to deal with.

4. Query Language

A query must be written in the format from which it is easy to extract the sub-queries so that data node can determine, whether it is able to perform some parts of the query in the pull method.

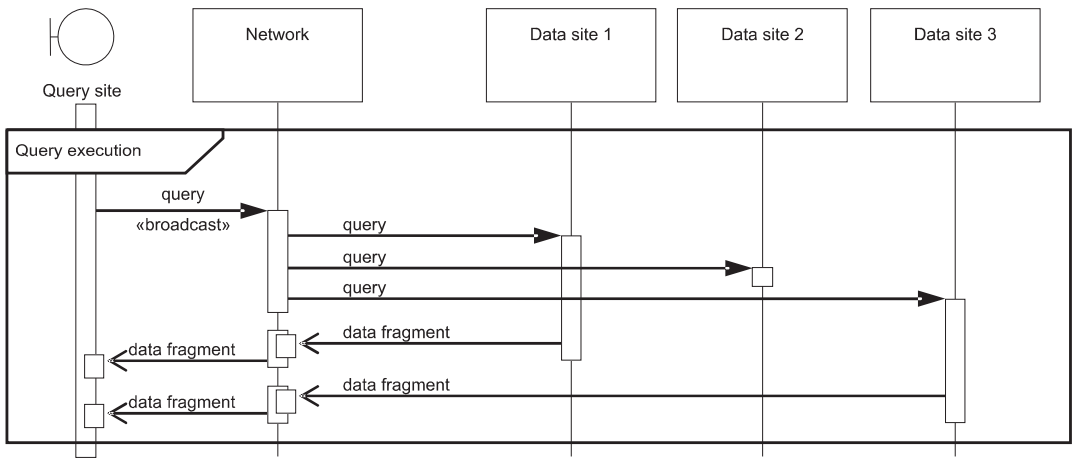

Fig. 2 Using broadcast message to send a query

The proposed system therefore uses object λ-calculus [3, 4], introduced in [6]. Object λ-calculus extends the classical λ-calculus by the possibility of working with objects and using relational algebra operations on object collections. Thus it retains the essential characteristics of classic λ-calculus and also allows the implementation of database operations. For extraction of sub-query in the object-λ-calculus the inverse operation to the β-reduction can be used.

Relational operations using the object λ-calculus can be written as follows [6]:

- access to an object property (attribute value)

```
object ◁ property
```

- selection operation

```
collection // (λ row | condition)
```

- projection operation

```
collection » {list of attributes}
```

- collection operation

```
collection » (λ row | condition)
```

- Cartesian product

```
collection1 × collection2
```

In addition, the proposed system introduces operations for the natural join operation:

- inner natural join

```
⋈ (collection1 × collection2)
```

- left outer natural join

```
⋈L(collection1, collection2)
```

- right outer natural join

```
⋈R(collection1, collection2)
```

- full outer natural join

```
⋈F(collection1, collection2)
```

For example SQL query

```
SELECT * FROM table WHERE id=10;
```

can be written using λ-calculus as:

```
table // (λ item | item◁ id = 10)
```

The query satisfies the first condition as any part can be executed separately.

5. Query Optimisation

With regard to rapid cluster structure changes the execution of query must be performed quickly. The query must be optimised in such manner so that data sites are able to process it in short time and have to transfer as little as possible data. The optimisation can be performed at two steps:

1. Global optimisation. It is performed at the query site. It tries to reorganise operations in the query so that amount of data sent back in responses is minimised.
2. Local optimisation. It is performed by every data site individually, which knows structure of local data by using Local Internal Schema and Local Conceptual Schema. It reorganises operations in query so that time required for execution is minimised.

The query site knows neither structure of the cluster nor organisation of data. It is therefore impossible to globally optimise queries for speed. The query site is unable to plan execution of subqueries for individual data sites or, in other words, to globally optimise amount of data. Unlike traditional DDBS it cannot even rely on replication of data and query for data from those sites which can access it most effectively. The query node can therefore base its optimisation only on knowledge of Global Conceptual Schema and knowledge of the query itself.

Secondly, data sites must not be expected to be able to access all the collections mentioned in the query. They are just expected to find out larger subquery they are able to execute and reply with the result of the execution. Operation with one collection (unary operation) can be executed if it has the collection accessible. However operations with two or more collections (multiary operations) are possible only if the data site can access all of the collections. Optimiser should therefore reorganise operations in such a way so that, if possible, unary operations are executed first, which minimises amount of data being processed by multiary operations.

We will illustrate the explained principles by following example:

```
⋈ (projects, workers) // (λ item | item◁projectName = DDBS)» {name, surname}
```

If a data site does have collection projects but does not have collection workers accessible, it is not able to perform join operation. It is therefore obliged to send everything from the collection projects even though the query site will later use only one record with the name DDBS.

However, by writing the query as follows:

```
⋈ (projects // (λ item | item◁projectName = DDBS), workers)» {name, surname}
```

the data site is able to perform the whole selection and would send through network only one record with the name DDBS.

Now imagine a different data site, which has only collection workers accessible. It has to response with the whole collection

workers even though the query site will, in the result, use only values of name and surname attributes.

By writing the query as follows:

```
⋈ (projects // (λ item | item◁projectName =
DDBS),workers » {name, surname})
```

the query site would not have enough information for join operation (it would lack information about attributes that should have been used – e.g. `project_id`). It implies that it is not possible to reorganise operations in an arbitrary manner. Every operation has to be executable even after optimisation which means that every required attribute has to be accessible. Moreover the resulting collection must not be changed either. Thus the optimisation has to follow these rules [3]:

- A unary operation cannot be moved in front of a multiary operation if it works with attributes of objects from several collections, which are used as parameters of the multiary operation.
- Unary operation collection cannot be moved. It transforms collection of objects of certain type to the collection of objects of different type. If the operation were moved in front of any other operation the new query would be even impossible to execute (because of collection item protocol change) or would be giving different results than the original one.
- Unary operation projection can be moved in front of binary operation join only in case that it returns all the attributes used by join.
- Projection can be moved in front of binary operations intersection, union and difference only in case that the list of operations is a subset of protocol of all original collections.
- It is always possible to move projection in front of binary operation Cartesian product.

6. Limitations of The New Architecture

The architecture introduced in this article does not intend to be a universal database system. It is a specialized solution for the implementation of distributed database systems in the dynamic networks environment. Therefore it has some problems which may prevent using the architecture in some situations.

It is impossible to do a lot of optimizations due to the fact that the GD/D does not exist. The sites do not even have access to the whole CD/D. Broadcast messages can also overload the network in case many queries are processed at the same time.

In most situations the querying sites do not have access to the whole database.

For the query site there is no way of how to find out whether it has already received all the data it has requested or if it has to wait longer.

It is impossible to provide referential integrity because the distributed database can be partially or even fully inaccessible.

This architecture allows just the read-only access to the distributed database. Destructive operations can be executed only locally on the data sites.

Poorly written queries can result in transfer of large amount of data and thus overload the network.

These limitations are results of the environment rather than the defects of the architecture. Thus the architecture can only be used when these problems are not limiting.

7. Conclusion

We can find many options how to use database systems in dynamic network environment. Ability to share public data stored in internal database of car [5] (e.g. information about parking places), to provide data specific to geographic location of the user (sharing maps) etc.

The present architectures are not suitable for such an environment. The proposed architecture removes their limitations; however it brings few of its own as a result of principles of dynamic networks.

In a short time a prototype of the system will be created so that its behaviour can be tested experimentally. It would be possible then to compare this new architecture with the existing ones.

Acknowledgement

This research was supported by the Centre of excellence for intelligent transport systems and services, ITMS 26220120028, University of Zilina.

ERDF - European Regional Development Fund

The project is being cofinanced by European Community

References

[1] DATE, C. J.: *An Introduction to Database Systems.* Addison-Wesley, 6th edition, 1995.
[2] HILLEBRAND, G. G., KANELLAKIS, P. C., MARISON, H. G.: *Database Query Languages Embedded in the Typed Lambda Calculus.* Proc. of 8th IEEE Symphosium on Logic in Computer Science, Montreal, Canada, 1993, pp 332-343.
[3] JANECH, J.: *Process Control in the Distribution Database,* PhD thesis, Faculty of Management Science and Informatics, University of Zilina, Zilina, Slovakia, 2010.
[4] JANECH, J., BACA, T.: Distributed Database Systems in Vehicular Ad-hoc Network. *Communications - Scientific Letters of the University of Zilina,* Vol. 12, 3/2010, pp 50-55.
[5] LIESKOVSKY, A., JANECH, J., BACA, T.: Data Replication in Distributed Database Systems in VANET Environment. *Proc. of 2nd International Conference on Software Engineering and Service Science,* Beijing, 2011, pp 304-307.
[6] MERUNKA, V.: *Object Modelling (in Czech).* Praha : Alfa, s.r.o., 2008.
[7] OSZU, M. T., VALDURIEZ, P.: *Principles of Distributed Database Systems.* Pearson Education, 2nd edition, 1999.
[8] SOKOLOVSKY, P., POKORNY, J., PETERKA, J.: *Distributed Database Systems (in Czech).* Academia, Praha, 6th edition, 1992.