STUDIA INFORMATICA 2011
Volume 32 Number 3B (99)

Marcin KARPIŃSKI, Jarosław KOSZELA
Military University of Technology, Informatics System Institute

OBJECT ORIENTED DISTRIBUTION IN MUTDOD

Summary. The paper contains overall description of distribution in object-oriented database created at Military University of Technology. This paper describes overall information about possible distribution's architectures and the ones provided in MUTDOD. The article contains information about distributed data storing, data processing and architecture of synchronization and replication units. Details about solutions provided in MUTDOD are also presented.

Keywords: distribution, query planner, replication, synchronization, MUTDOD

OBIEKTOWO ORIENTOWANE ROZPROSZENIE W MUTDOD

Streszczenie. Artykuł zawiera ogólny opis architektury rozproszenia w obiektowej bazie danych MUTDOD, która jest tworzona w Wojskowej Akademii Technicznej. Artykuł opisuje możliwe rozwiązania architektoniczne wraz z rozwiązaniami przewidzianymi w MUTDOD. Artykuł zawiera informacje o modułach synchronizacji i replikacji oraz rozwiązaniach zastosowanych przy rozproszonym przechowywaniu i przetwarzaniu danych.

Słowa kluczowe: rozproszenie, planowanie zapytań, replikacja, synchronizacja, MUTDOD

1. Introduction

Military University of Technology Distributed Object Database (MUTDOD) is a new look for storing and manipulating data. Nowadays systems usually need to process large amount of data. Simple machines with standard database systems are slowly reaching their limits. For the purpose of processing large amount of data people have created computing clouds and clusters. Unfortunately both of them are not perfect solutions for processing billions of records. Clouds are usually beyond the control of organization. Their units are con-

trolled by a third party company, what eliminates problems with scalability, but reduces awareness of how crucial data are stored and processed. Usage of self controlled cluster is also possible but it needs a lot of management. Even if easily manageable, cluster of relational databases still does not solve one problem – the system is probably written using object oriented paradigm whereas database is relational. It provides necessity for efficient ORM[1] [1, 3]. Database which can store data in a way that object oriented system understands (so store data in object form) and is capable of processing data on multiple units when it is possible, would eliminate necessity for ORM solution and provide a tool for faster data processing. MUTDOD is an example of such a database. Not only is it able to perform as a single unit, but it can also operate like a cluster. It stores not only attributes or properties of objects but also its methods, and in addition, MUTDOD is able to execute stored methods in a distributed environment.

2. Central Server vs. P2P

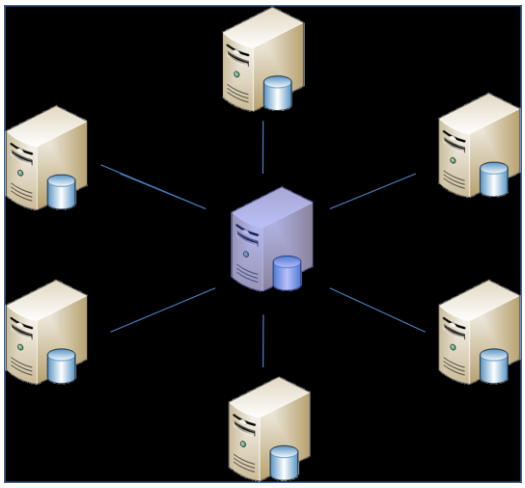

Fig. 1. Central Server Architecture
Rys. 1. Architektura z serwerem centralnym

MUTDOD architecture is based on two different, but similar in some aspects, ideas. MUTDOD was design as a single database unit which also has to be able to operate as an element of a distributed environment [9, 10]. Natural way of meeting that requirement would be central server architecture. This type of architecture has got some serious disadvantages which rather eliminates it from using with crucial data. First and the most important disadvantage is the central unit. Its failure causes a fail of the whole database, and crucial data becomes unavailable. Moreover central unit is a gateway for all communication, so it has to process all incoming requests, what causes slowdown of whole system or even fail of system

[1] ORM – object relational mapping is necessary because of impedance mismatch

when machine is no longer able to process such a big number of connections. P2P architecture is more efficient in such systems, but it causes other significant problems. First of all, there is no central point for all types of blockades, and controlling. Performing transaction in such environment is very complicated because before anything will happen, blockades on all machines have to be set up. There are also serious timing problems. There is no simple solution for situation where two equal in priority (because in P2P architecture we treat all machines equally) requests of data manipulation are received. Which one should be executed at first place? P2P architecture has high risk of deadlocks and starvation. Finally all information has to be populated among all devices to keep database in consistent state.

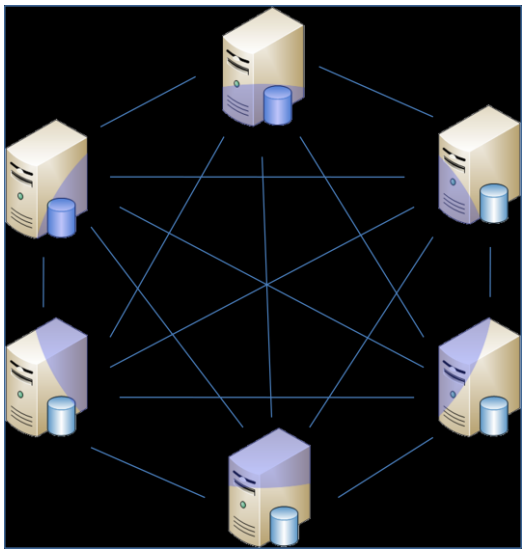

Fig. 2. P2P Architecture
Rys. 2. Architektura P2P

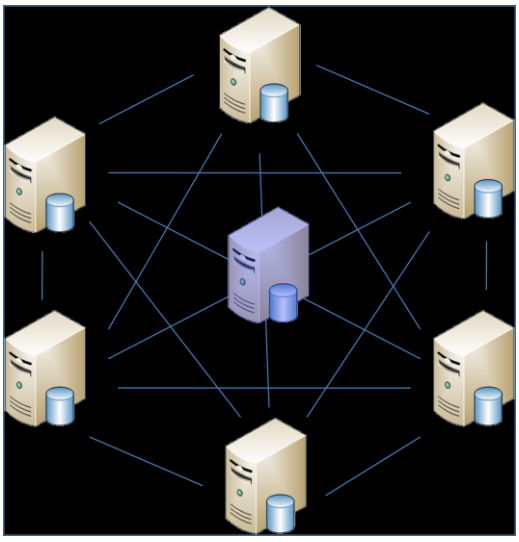

Fig. 3. Hybrid Architecture
Rys. 3. Architektura hybrydowa

The third possible solution is a hybrid architecture. It is a unique mix of characteristics of both already mentioned architectures. Hybrid architecture [5] is simply a central machine in a standard peer-to-peer environment. Such add-on to P2P environment solves the problem of timing and synchronization but unfortunately brings back another one – weak point of central server. Besides all this advantages and disadvantages of each described architecture there is one more important characteristic which we should consider – transparency [8]. Transparency is a feature which we can apply on all levels of database structure. First of all, user should not be aware of database structure. The fact that user is working with cluster rather than with a single node should be invisible for him. When users are working with database they cannot be aware of database structure changes. They cannot be forced to reconfigure the client software, because some additional nodes joined the cluster, or because additional data replicas are now available. Information about distributed data processing [10] (so that user request is processed by more than one node) should not be visible to users[2]. User should not be aware of data localization, fragmentation, data processing details so number of nodes, their localization, used for query execution. MUTDOD system was designed to provide transparency on all available levels [3, 7, 11].

3. Federation vs. Election

MUTDOD system is capable of working in two architectures: hybrid and P2P. The main idea of MUTDOD architecture is distinguishing node types. Data nodes and management nodes cooperate in environment, but they are not necessary on the same machines. Of course, if a node does not have both data server and management server it is not capable of performing as a single unit or single database. Both hybrid and P2P has one single point of start – metadata replication. Mode switching depends on configuration of a database. If all nodes store replicas (up-to-date replicas) of metadata, mode can be switched in a real-time. If some nodes do not have up-to-date metadata, replication has to be performed before mode switching. MUTDOD is even capable of working in semiP2P architecture, so where a number of management servers is running but not all data servers have dedicated ones.

While working in P2P architecture MUTDOD database acts like a federation of single units. They can exchange data on-the-fly or perform periodical updates. There is also a mode in which when operations causing synchronization problem occur all of them are rolled back and before they are executed once more the system decides the order of requests. All of pro-

[2] Only when it is not important for user – user is able to manipulate query execution by using new keywords available in DDQL [6]

vided mechanism cause some additional network transfer and some delays caused by the synchronization operations.

On the other hand we get simple hybrid architecture. MUTDOD hybrid architecture has got significant advantage over standard version – central server is not appointed, system decides which one should be at that moment the central one. During creating a cluster the administrator decides which one is the chosen one, and also sets up which node is its deputy. When one of these nodes is going down another performs reelection and finds its substitute. In this case whole database is protected from single node failure.

Which mode is better? It is hard to say. Both modes have got advantages and disadvantages, the cost of peer2peer mode is amount of data transfer required to set up blockade on objects, transfer required to loop back from starvation or deadlocks. In election these problems do not occur but there is a weak point that all requests have to be passed through the central unit. Database administrator or designer should choose which one is more suitable for his needs.

4. Data replication

The core idea of distributed systems is processing data by more than one machine. This is a natural way of improving system performance. However before distributed processing will be possibly we have to take care of one more thing. Before query can be processed by more than one node all nodes have to have the necessary data. The process of coping data from one unit to other is called replication. There are three types of replication – snapshot replication, merge replication and transactional replication. [14]

In the situation described above we have to use the first type of replication – snapshot replication. This type of replication is, as name suggest, similar to taking a snapshot of data available on one unit and transferring it to different machine. This is not only starting point for parallel data processing but also a way of protecting data. If the first node fails we still have access to data on the second one. But this is unfortunately also starting point for some troubles. In relational databases possessing more than one version of data is called redundancy and is usually something we should avoid, because it is potentially a perfect way to reach database inconsistency. Having the same information saved in more than on record[3] causes the situation in which you have to modify all records when data needs to be changed. If only one copy will not be updated on time the database will be in inconsistent state, and resolving which records is up-to-date will be impossible. The process of keeping database in consistent state by updating all copies of the same data is called transactional replication or data syn-

[3] In relational databases, and in more than one object in object-oriented

chronization. Unfortunately there is one more problem – what if second unit already has some data, and what if some of his data is a copy of data from first node?

To solve this problem, the database (or any distributed environment) has to be able to perform merge replication. Merge replication is a process of synchronizing data between two or more nodes. Units exchange data to achieve state in which nodes are copied of each other so any node has the data from all other nodes. This type of replication has to solve problem with two versions of the same data. If first node posses the same data that the second node but in different version, we have to find out which version is up-to-date.

Merge replication in MUTDOD system uses simple mechanism for solving such problem. Administrator or database designer has to choose which version should be used. In MUTDOD system we have built in some possible solution for this problem, so algorithms in which the version problem is solved by using nodes priority. For example one algorithm always uses the version from node that is joining the system, whereas second algorithm always uses the version that is already in the system. Moreover, MUTDOD system offers database designer the possibility to design and implement own algorithm and use it to solve this problem. At this point administrator can implement his own comparing algorithm using C# language and built in IComparer interface.

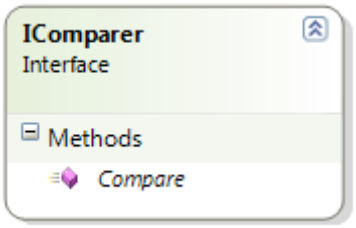

Fig. 4. IComparer Interface
Rys. 4. Interfejs IComparer

As was mentioned above, having more than one copy of the same data causes situation in which nodes have to be synchronized when data is being updated. Transactional replication algorithm has to use all advantages of system architecture because this replication runs constantly and can produce large transfer between nodes.

Transactional replication algorithm available in MUTDOD system is simple and effective considering both possibly architectures. Designed algorithm is based partially on previous works on solving that problem [13]. In many present databases the transactional replication is performed by choosing one node which will be a gateway for all modifications. Such node will have to follow all modification and localization of all objects and perform all necessary operation to keep database in consistent state. This approach however has one security risk – node failure[4]. MUTDOD system algorithm is based on this approach but it dismisses the risk. When an update operation (so any operation which changes data) is being performed central server (or

[4] The same situation as with central-server architecture

management node to which client is connected) is choosing the node which will be the gateway only for this particular operation. The node choosing algorithm may work in many different ways, not only designer can implement his own one, but also load balanced one can be used. This node is responsible for updating data on all nodes that have a copy of modified data. To speed the process data distribution is divided. The whole process is shown in figure 5.

An important thing about the mechanisms of replication in MUTDOD is a fact that they have to consider fragmentation [4] of data. Data can be fragmented both in vertical and horizontal way [13]. In object-oriented environment this two types of fragmentations [12] are very similar. To distinguish these types in MUTDOD vertical mode stores two objects on different nodes. Horizontal division is based on distinguishing parts of object and setting individual OID number for these parts.

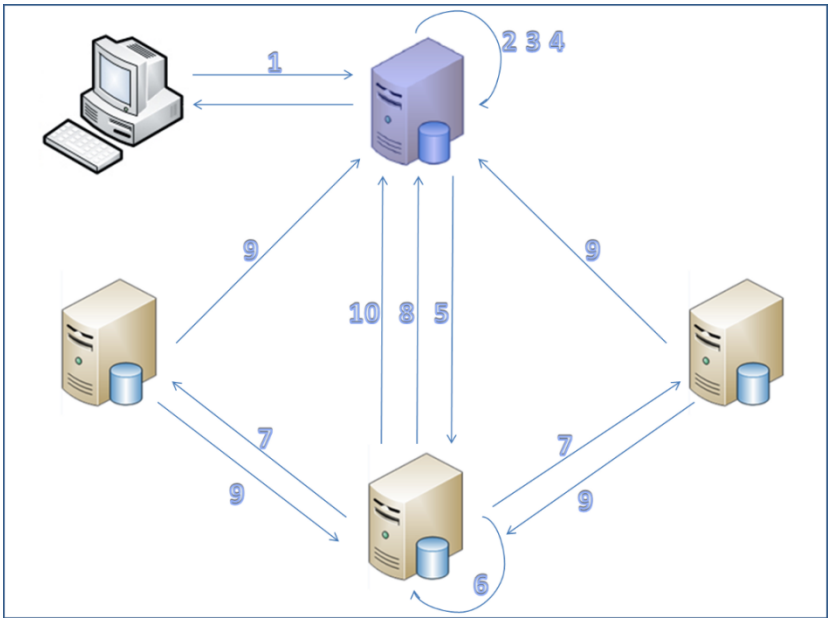

Fig. 5. Data population algorithm
Rys. 5. Algorytm populacji danych

Client sends a query to the management node.

1. Management server starts the query execution.
2. All necessary objects are being blocked.
3. Management server chooses the node which will perform the operation – in case of add operation the server which should carry replicas are also chosen.
4. Server passes necessary data to the chosen node.
5. The node updates its own data.
6. Node starts the operation on other nodes.
7. Node sends back the information about successful update.
8. Other nodes update their data and send information to the management server and chosen node.

9. Information about successful replication of all nodes reaches the management server and can be replicated to other management servers (if more than one exists).

5. Distributed processing

When all necessary operations of data replication are performed we can now start dividing the work of executing the query. To divide the work for all nodes we have to use powerful distributed query planner module [2]. This module has to answer few questions. First of all we have to find out if the query is worth dividing. Let us consider a simple query which sums two values. We can easily say that this should be executed immediately on the node to which client is connected, because the time and CPU power needed to perform planning operation are much bigger than simple executing this operation.

The second question is – is it possible to divide query? We can think about situations when one server has all data that query needs. In such situations there is no point dividing the work because only one unit is able to execute the query. After answering this two main questions division can be perform.

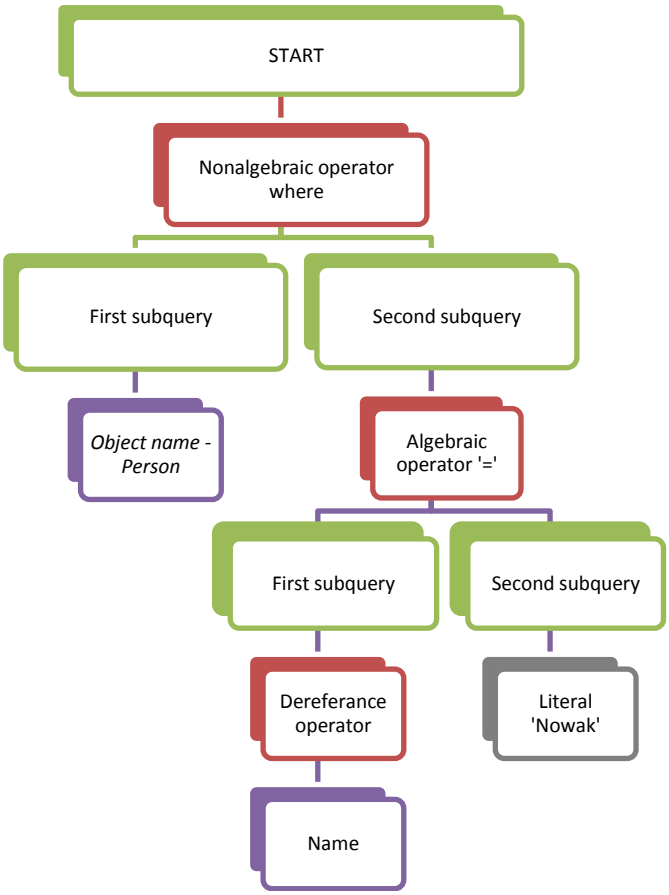

Fig. 6. Token tree
Rys. 6. Drzewo Tokenów

Process of dividing work for all nodes is based on a token tree [15]. This process has two steps. First step is trying to divide work into number of nodes, based on load of nodes and data they posses. The result of division is a plan of query in a form of number of queries and information in what order the queries should be executed and how their results should be mixed. Sometimes perfect division cannot be done and some queries need to be executed in certain order. After this is completed the second step is performed. The token trees of subqueries are analyzed, and in case the machine has got more than one CPU unit[5] the division for CPUs is performed.

6. Other consequences of distribution

Distribution has far more consequences then these described in this article. As such consequences we should take into account such things as query language modifications [6] necessary to manipulate distribution like data partitioning or localization or some performance switches. Also, the metamodel [11] needs to be adapted to distribution requirements. The data about localization and partition of each object has to be saved and kept somewhere – so metamodel has to consider storing such things. The last thing is data fragmentation. Metamodel, language like the distribution module itself has to be able to work with partitioned data both vertically and horizontally.

7. Summary

As you can see MUTDOD system is something more than a simple cluster. MUTDOD system is able to work both as single unit database and a cluster with both P2P and central management architecture. MUTDOD system covers all problems connected to distribution – user is not even aware that he is working with more than one unit. MUTDOD system is trying to be as transparent as it is available on all levels – both architecture, structure, processing etc.. MUTDOD conveys a new look at database systems, which perfectly fits present trends of cloud computing and which can show directions of database evolution. Military University of Technology will continue working on MUTDOD and on finding best FAR strategy for data.

[5] Or more than one core

BIBLIOGRAPHY

1. Brzozowska P., Góralczyk M., Jesionek Ł., Karpiński M., Kędzierski G., Kędzierski P., Koszela J., Wróbel E.: System obiektowy = obiektowa baza danych + obiektowa aplikacja. Studia Informatica, Vol. 31, No. 2B (90), Gliwice 2010.
2. Jesionek Ł.: Projekt generatora planów wykonywania zapytań dla obiektowej bazy danych. Praca dyplomowa WAT, Warszawa 2010.
3. Góralczyk M.: Projekt oprogramowania zarządzającego obiektową bazą danych. Praca dyplomowa WAT, Warszawa 2010.
4. Karpiński M.: Projekt mechanizmu replikacji i synchronizacji elementów obiektowej bazy danych. Praca dyplomowa WAT, Warszawa 2010.
5. Coulouris G., Dollimore J., Kindberg T.: Distributed Systems: Concepts and Design. Addison Wesley Longman, 2005.
6. Brzozowska P.: Projekt analizatora syntaktycznego i semantycznego obiektowej języka zapytań. Praca dyplomowa WAT, Warszawa 2010.
7. Wróbel E.: Projekt interfejsu programistycznego dostępu do obiektowej bazy danych. Praca dyplomowa WAT, Warszawa 2010.
8. Tanenbaum A. S., Steen M.: Systemy rozproszone Zasady i paradygmaty. WNT, Warszawa 2006.
9. Date C. J.: Twelve rules for a distributed database. Computer World 21(23), 1987.
10. Orfali R., Harkey D., Edwards J.: The Essential Distributed Objects Survival Guide. John Wiley & Sons, 1996.
11. Koszela J., Góralczyk M.: Architecture of object database, in progress.
12. Malinowski E.: Fragmentation Techniques for Distributed Object-Oriented Databases. University of Florida, 1996.
13. Sikorska M.: Praktyczne porównanie mechanizmów 2PC i 3PC. Wydział Inżynierii Mechanicznej i Informatyki Politechniki Częstochowskiej, 2006.
14. Garcia-Molina H., Ullman J. D., Widom J.: Systemy baz danych. WNT, Warszawa 2006.
15. Date C. J.: Wprowadzenie do systemów baz danych. WNT, Warszawa 2000.

Recenzenci: Dr inż. Ewa Płuciennik-Psota
Dr inż. Aleksandra Werner

Wpłynęło do Redakcji 16 stycznia 2011 r.

Omówienie

Projekt MUTDOD, czyli Military University of Technology Distributed Object Database, to rozproszona, obiektowa baza danych, działająca w sfederowanym modelu rozproszenia. Projekt ten, tworzony w Wojskowej Akademii Technicznej, ma przede wszystkim rozwiązać problem impedancji, przez co ma stanowić idealną platformę dla systemów opartych na paradygmacie obiektowości.

We współczesnych systemach spotykamy się z sytuacją, kiedy obiekty systemu muszą być transformowane na tabele relacyjnej bazy danych. Aby rozwiązać ten problem, należałoby przechowywać obiekty bez zmiany ich formy, np. bez zbędnej konwersji do postaci tekstowej.

We współczesnym świecie istnieje także tendencja do przechodzenia na technologie Cloud Computing, czyli rozproszonego przetwarzania danych. MUTDOD ma oferować podobne możliwości, tj. umożliwiać zarówno prace z jednym węzłem, jak i z wieloma. O ile baza będzie dysponować więcej niż jednym węzłem, system ma umożliwiać zarówno rozpraszanie danych, jak i obliczeń na wszystkie węzły. Jednym z założeń sytemu MUTDOD było ograniczenie pracy administratora i projektanta do minimum przy rozpraszaniu przetwarzania. Sam proces przetwarzania ma być jak najbardziej bezobsługowy, tak aby maksymalnie odciążyć programistę.

Ważnym elementem systemu jest język zapytań. Obecnie wykorzystywane języki zarówno deklaratywne, jak i imperatywne nie do końca spełniają wymagania systemu MUTDOD. Aby programista czy administrator miał możliwość manipulowania procesem rozpraszania, niezbędne jest wprowadzenie do języka dedykowanych temu zadaniu form składniowych, które umożliwiłyby użytkownikowi nie tylko oznaczenie instrukcji, które powinny być zrównoleglone, ale również kontroli nad automatyczną wersją tego procesu. Cały czas jednak należy pamiętać, że rozmieszczenie obiektów znacząco wpływa na możliwość rozpraszania. Oba te fakty powodują, że system MUTDOD wymaga języka posiadającego szczególne cechy, co przekłada się na konieczność zaimplementowania własnego języka.

MUTDOD jest systemem bazodanowym nowej generacji. Może on pracować zarówno w trybie prostej, jednomaszynowej bazy danych, jak również jako sfederowany system multiwęzłowy zdolny do przetwarzania i przetrzymywania rozproszonych danych. Tryby pracy z serwerem centralnym bądź też w wersji pełnego P2P, a także możliwość definiowania własnych algorytmów rozmieszczania obiektów, bądź ich porównywania daje administratorom i programistom całkowitą kontrolę nad działaniem systemu.

Addresses

Marcin KARPIŃSKI: Wojskowa Akademia Techniczna, Wydział Cybernetyki,
ul. gen. Sylwestra Kaliskiego 2, 00-908 Warszawa, Poland, marcin@karpinski.waw.pl.
Jarosław KOSZELA: Wojskowa Akademia Techniczna, Wydział Cybernetyki,
ul. gen. Sylwestra Kaliskiego 2, 00-908 Warszawa, Poland, jkoszela@wat.edu.pl.