14th International Conference
MIXED DESIGN
MIXDES 2007
Ciechocinek, POLAND
21 – 23 June 2007

QUERY OPTIMIZATION IN GRID DATABASES

J. MURLEWSKI, T. KOWALSKI, R. ADAMUS, B. SAKOWICZ, A. NAPIERALSKI
TECHNICAL UNIVERSITY OF LODZ, POLAND

KEYWORDS: Query optimization, Grid architecture, SBA, Data-intensive grid

ABSTRACT: This paper discusses methods for query execution and optimization in object-oriented grid databases. The queries for distributed databases have become even more complex. This cause a difficulty for query execution and optimization process, which requires advanced algorithms and techniques to reduce processing and communication cost. This paper gives the outline of distributed data processing, which is now the subject of our study and development.

INTRODUCTION

The query optimization techniques for grid databases have been comprehensively studied for relational systems [1,2]. However, those methods are not for a general purpose and not all of them are appropriate for object-oriented distributed databases. Moreover, optimization techniques for object-oriented gird databases have been under intense development, but they still lack dedicated algorithms due to the complexity of the underlying schema, an irregularity of query languages, a large space of possible execution plans etc.
The query optimizations are essential in grid databases as it must produce a query execution plan including an access path, join and fetch order strategies together with distribution of data on servers.
The naive approach to the execution of distributed query might be to fetch by a client all required data from remote nodes and then execute the query locally. Nevertheless this approach would result in fetching huge amounts of data, provided that some of them might not even be required. One solution to this problem is to reduce the amount of data transmitted that is necessary to execute the query.
The main aim of distributed query optimization is to minimize:

- the communication cost,
- the overall query execution time,
- the CPU usage and IO cost at the client side,
- the CPU usage and IO cost at the grid nodes,
- the response time.

The main optimization strategy is to reduce the communication cost, which is the most important factor and most strategies are devoted to reduce the size of intermediate results (e.g. semi-join [7]).

During the query optimization process there are varieties of methods that can be applied, such as:

- semi join;
- independent subqueries;
- indices;
- redundancy;
- query rewriting.

The query optimization in grid databases is even more complex because of the large search space of possible execution plans, which is even more difficult in a case where nodes can interact with each other.
On the other hand, the advantage of grid databases is the possibility of the parallel query execution, which might extensively enhance the performance.
The query optimization engine requires also designing a cost model to accomplish listed above requirements, which is a hard issue considering the diversity of underlying databases, different schemas, varying communication lines, availability of indices etc.
In this paper we consider optimization techniques for Stack-Based Approach ([3], [7]), which is the framework for our development.

THE GRID ARCHITECTURE

The architecture of a data grid is presented in Fig. 1. The main part of the grid is a *global virtual object and service store*, which manages virtual objects and provides services. A global virtual store is defined through updateable views [5, 6]. It is worth noticing that the views in Stack-Based Approach are just complex objects, thus they can contain methods and procedures as well as local variables that store the state of these views.

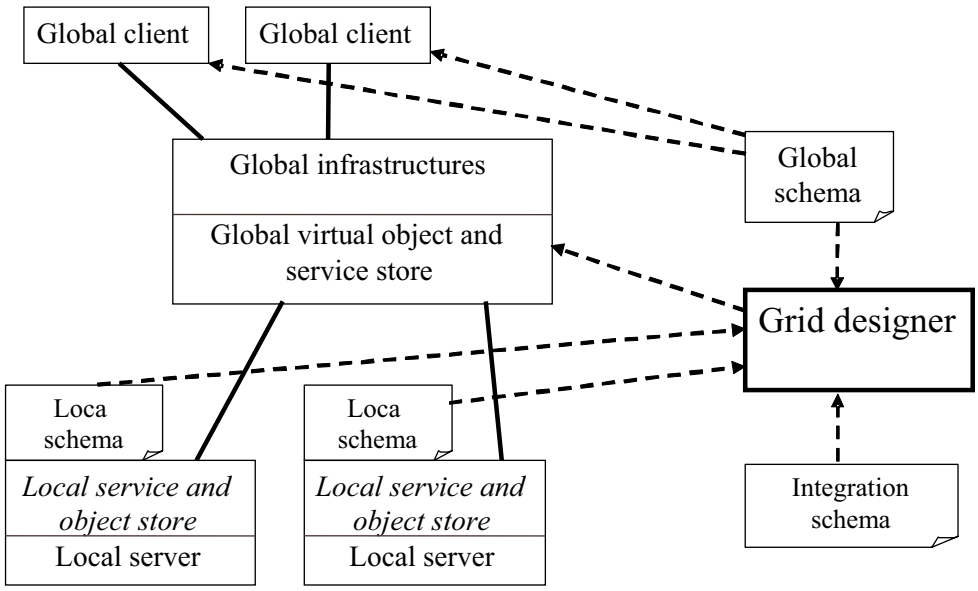

Fig. 1. Grid architecture.

Global clients are applications or other services which interact with **global infrastructure** (e.g. retrieve, update, and insert virtual objects).
From the query optimizer point of view a *global schema* is especially important part of the architecture as it defines objects and services provided by the grid. The global schema plays a crucial role during building the execution plan as it contains schemas integration information, provides details about indices and replicas.
The grid offers services and data of *local servers* to these global clients. A *local schema* defines data and services provided by a local server. These schemas can be different at each local server and are invisible to the grid users. However the local schema is not utilized during global query optimization. The *integration schema* contains additional information how the data and services of local servers are to be integrated into the grid.
The solid lines represent runtime relationships i.e., queries, requests, and responds. Whereas, dashed arrows illustrate association that are used during development of the grid.

LOCAL VS. GLOBAL OPTIMIZATION

When dealing with grid databases it is essential to make a clear distinction between a local and global query optimization.
The local query optimization is applied only for single database node and is not driven by a global optimization. Local query optimization and execution retains inclusive control over local processing and choosing the best execution strategy (e.g. choosing join order, usage of indices, rewriting, pushing expensive operators down an execution tree).
Global query optimization on the other hand, drives the query execution in the distributed environment and consists of many phases. Firstly, the query is parsed and decorated with metadata that describes a localization of data in the grid. Secondly, during the optimization phase the query might be rewritten and some optimization techniques might be applied, such as factoring out independent subqueries, pushing expensive operators down the query tree, removing dead subqueries etc. The query optimizer decides also how the query should be decomposed into subqueries addressed to nodes in the grid. After this step, the query optimizer generates possible execution plans, among which the best plan is chosen from a point of an execution cost view. According to the generated execution plan the subqueries are sent to appropriate servers. Finally, the result must be assembled from partial results, which were returned by subqueries. It is worth noticing that the global query optimization might not know in advance the actual execution cost of each subquery on local nodes. However, the global optimizer should have at least basic statistics about local servers such as cardinality of dataset, average size of fields, communication cost etc.

QUERY OPTIMIZATION STRATEGIES

The query execution in distributed environment generally is composed of the following steps:

1. query parsing,
2. data location,
3. global query optimization,
4. query decomposition into subqueries addressed to local nodes,
5. the subqueries are sent to local nodes in an appropriate order,
6. results of subqueries are collected and combined to make the final result.

There are, however, some strategies of optimization in grid databases that vary in scalability, performance and complexity.

Client side processing strategy

In this strategy, the client fetches all the data required from nodes. After receiving all the data, it starts to execute the query locally. Even thought, this strategy is straightforward and easy, it has many drawbacks. One of its disadvantages is related to mandatory fetching huge amounts of data, even thought small subset of them might be sufficient to build the final result. This leads to huge cost of data transmission and an extensive query execution time. Moreover, clients must be equipped with memory, storage and computational power capable of processing the query efficiently.

Static decomposition

This approach is more advanced than the previous one. During this strategy, the query is decomposed into subqueries (in the Fig. 2 they are marked as Q1, Q2 and Q3), which are sent to nodes simultaneously.

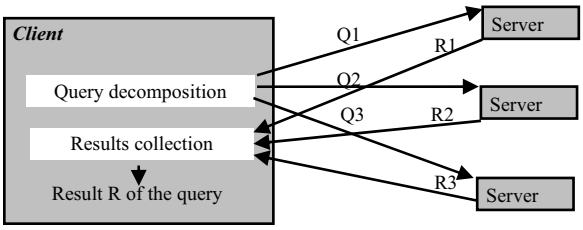

Fig. 2. Static decomposition.

The optimization of subqueries must comply with the fact, that only data required to constitute final answer should be fetch. The advantage of this strategy is efficiency for horizontal data fragmentation. However, this strategy is inefficient for more advanced and complicated queries and those containing dependant subqueries, where result of one query depends on other subqueries.

Example of static decomposition

Consider the following query, which can be statically decomposed into subqueries addressing servers in distributed environment.
The following query expressed in SBQL language list all students:

Student . name (1)

After decoration the query with servers locations:

Student {A,B} . name (2)

The student objects are distributive with respect to horizontal fragmentation and are located at servers A and B.

(Student{A} union Student{B}). name (3)

As long as the dot operator is distributive [5] with respect to horizontal fragmentation the query might be rewritten as follows:

Student{A} . name union Student{B} . name (4)

The two subqueries might be now executed simultaneously and a client can combine results.

Dynamic decomposition

This technique differs from the previous one in a way the subqueries are generated and the order they are sent. During the dynamic strategy, the query optimizer decomposes the query Q (Fig. 3) into subquery Q1, which is sent to server server1. After receiving the result R1 of the first subquery, the obtained result is used to generate another subquery Q2, which is sent to another server server2. After receiving the result and combining it with previous results, it is possible to generate another subquery addressing next server. This process continues until final result is obtained.

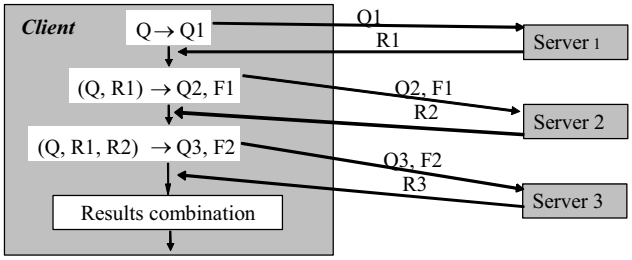

Fig. 3. Dynamic decomposition

This approach is conceptually more advanced than previous one. However, it lacks the possibility of parallel query execution. The advantage of this technique is better optimization of subqueries by early reduction of the size of intermediate query results.

Example of dynamic decomposition

An example on dynamic decomposition includes a subquery, which drives the execution of an outer query. The following query lists all students that are older than student named Smith:

Student where age > ((Student where name = "Smith").age) (5)

The student objects are distributive with respect to horizontal fragmentation and are located at servers A and B. After decorating the query with server locations:

Student {A,B} where age > ((Student {A,B} where name = "Smith").age) (6)

In this example the inner query must be executed first, so that it result can be used to execute the outer query. Firstly, the age of Smith must be found and it is expected that this information must be stored at one of the servers A or B.

A:= (Student {A,B} where name = "Smith").age (7)

Because this query contains distributive operator *where*, it can be rewritten into the following query:

A:= (Student{A} where name = "Smith").age union (Student{B} where name = "Smith").age (8)

It is worth noticing that the previous query can be executed in parallel at servers A and B.
Finally, as the age of Smith is obtained, it is possible to execute the outer query:

Student {A,B} where age > A (9)

As this query contains distributive operator where, it can be rewritten into the following query:

Student {A} where age > A union Student {B} where age > A (10)

This example presents a dynamic decomposition of a query, which can be dynamically decomposed in to subqueries according to the result of the inner subquery.

Hybrid decomposition

The most promising strategy is hybrid decomposition, which combines the advantages of static and dynamic decompositions. This approach is very similar to semi-join technique [7]. Moreover this strategy allows shipping intermediate results along with subqueries to servers in order to improve performance and communication lines utilization. The main idea of this optimization strategy is to generate subqueries, which might be sent together with intermediate results simultaneously to many servers. This process is repeated until final result is obtained. This strategy seems to be the best for simple and more complex queries. Moreover, the hybrid decomposition allows equal servers and transition lines utilization. On the other hand it is the most conceptually difficult and requires advanced cost model optimizer module.

CONCLUSIONS

In this paper an overview of query optimization techniques in grid databases is presented.
The future work will focus on developing cost models for static, dynamic and hybrid strategies of query execution. As it was described the optimization process is highly dependant on chosen optimization strategy, which makes the whole process more complex. The development of the grid framework is based on the Stack-Based Approach. The presented strategies are being under development of prototype framework.

THE AUTHORS

Jan Murlewski, Bartosz Sakowicz, Andrzej Napieralski are with the the Department of Microelectronics and Computer Science, Technical University of Lodz, Poland
E-mail: murlewski@dmcs.pl
Tomasz Kowalski, Radosław Adamus are with the Computer Engineering Department, Technical University of Lodz, Poland

REFERENCES

[1] M.T.Özsu, P.Valduriez: Principles of Distributed Database Systems, Second Edition, Pren-tice-Hall 1999.
[2] D.Kossmann: The State of the Art in Distributed Query Processing. ACM Comput. Surv. 32(4): 422-469 (2000).
[3] Subieta K. 2004: Teoria i konstrukcja obiektowych języków zapytań. Wydawnictwo PJWSTK, Warszawa 2004, ISBN 83-89244-28-4.
[4] Płodzień J., Subieta K. (alias A.Kraken). Object Query Optimization in the Stack-Based Approach. Proc. of 3rd ADBIS Conf., Maribor, Slovenia, 1999, pp.303-316, Springer LNCS 1691
[5] H.Kozankiewicz, J.Leszczyłowski, J.Płodzień, K.Subieta. Updateable Object Views. Institute of Computer Science, Polish Academy of Sciences, Report 950, Warsaw, Poland, 2002
[6] H. Kozankiewicz, K. Stencel, K. Subieta: Implementation of Federated Databases through Updatable Views. Proc. of the European Grid Conference, Amsterdam, The Netherlands, 2005, LNCS
[7] P.A.Bernstein, N.Goodman, E.Wong, C.L.Reeve, J.B.Rothnie Jr.: Query Processing in a System for Distributed Databases. ACM Trans. Database Syst. 6(4): 602-625 (1981)