A FRAMEWORK FOR PARALLEL QUERY PROCESSING ON GRID-BASED ARCHITECTURE

Khin Mar Soe, Than Nwe Aung, Aye Aye Nwe
University of Computer Studies, Yangon, Myanmar

Thinn Thu Naing, Nilar Thein
University of Computer Studies, Yangon, Myanmar

Keywords: Grid computing, parallel query processing, distributed query processing, query optimization, dynamic programming.

Abstract: With relations growing larger, distributed, and queries becoming more complex, parallel query processing is an increasingly attractive option for improving the performance of database systems. Distributed and parallel query processing has been widely used in data intensive applications where data of relevance to users are stored at multiple locations. In this paper, we propose a three-tier middleware system for optimizing and processing of distributed queries in parallel on Cluster Grid architecture. The main contribution of this paper is providing transparent and integrated access to distributed heterogeneous data resources, getting performance improvements of implicit parallelism by extending technologies from parallel databases. We also proposed the dynamic programming algorithm for query optimization and site selection algorithm for resource balancing. An example query for employee databases is used throughout the paper to show the benefits of the system.

1 INTRODUCTION

Today, both commercial and scientific applications increasingly require access to distributed resources. Where there is more than one database supported within a distributed environment, it is straightforward to envisage higher-level services that assist users in making use of several databases within a single application. The Grid is an ideal environment for running applications that need extensive computational and storage resources (Oldfield, R., Kotz, D., 2001). Grid technologies facilitate efficient sharing of data and resources in a heterogeneous distributed environment. The need to reduce response time is evident in decision support applications in which human beings pose complex queries and demand interactive responses.

There are perhaps two principal functionalities associated with distributed database access and use. They are distributed transaction management and distributed query processing (Smith, J., Gounaris, A., Watson, P., Norman W., Alvaro A.A., Sakellariou, R., 2003). This paper is concerned with parallel query processing on the Grid which describes a prototype infrastructure for supporting parallel query optimization and evaluation within a Grid setting. In Grid environment, finding the answer of a user query will require translating it into internal representations, structuring optimal query, splitting it into parts, retrieving the answers of these parts from remote nodes, and merging the results together to calculate the answer of the initial query.

This paper presents an approach to parallel query processing on Grid system. Query scheduling including processor allocation and optimization that support intra and inter-query parallelism for read-only queries are discussed. The remainder of this paper is organized as follows.

In Section 2, we briefly describe the related work in parallel execution of queries in a Grid environment. In Section 3, we present some background theories about parallel query processing. In Section 4, we present our design of the system. We conclude our system in Section 5.

Mar K., Nwe Aung T., Aye Nwe A., Thu Naing T. and Thein N. (2005).
A FRAMEWORK FOR PARALLEL QUERY PROCESSING ON GRID-BASED ARCHITECTURE.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 203-208
DOI: 10.5220/0002555202030208
Copyright © SciTePress

2 RELATED WORK

As stated in (Andrade, H., Kurc, T., Sussman, A., and Saltz, J., 2001)(Beynon, M. D., Sussman, A., Catalyurek, U.,. Kurc, T., and Saltz, J., 2001)(Dail, H., Sievert, O., Berman, F., Casanova, H., YarKhan, A., Vadhiyar, S., Dongarra, J., Liu, C., Yang, L., Angulo, D., Foster. I., 2003), several research projects have been developed distributed data management over the Grid. The Active Proxy-G service (Alpdemir, N., Mukherjee, A., Paton, N. W., Watson, P., Fernandes, A. A. A., Gounaris, A.. and J. Smith., 2003) is dedicated to be able to cache query results, use these results for the parts of a query that cannot be produced from the cache, and submit the sub-queries for final processing at application servers that store the raw data sets. (Rodr´ guez-Mart´ nez, M., Roussopoulos., N., 2000) proposed a database middleware (MOCHA) which is designed to interconnect distributed data sources.

Many types of environments for executing grid-aware applications can be found in the literature. (Gounaris, A., Norman W., Alvaro A.A., Sakellariou, R.)(Tierney, B., Johnston, W., Lee, J., Hoo, G., Thompson, M). In distributed database systems, there is an infrastructure that supports the deliberate distribution of a database with some measure of central. In federated database systems, we can see the systems that allow multiple autonomous databases to be integrated for use within an application, and in query-based middleware, there is a query language that is used as the programming mechanism for expressing requests over multiple wrapped data sources. This paper is most closely related to the third category, in that we consider the use of parallel query processing for integrating various Grid resources, including database systems.

3 PARALLEL QUERY PROCESSING

Parallel query processing is a well established mechanism in relational DBMS. The objective of parallel query processing is to translate a high-level query into an efficient low-level execution plan and allocate processors to each operation in such a way that the overall query execution time is minimized. (Kossmann, D., 1998)(Lu, H., Shan, M-C., Tan, K-L., 1991)(DeWitt, D.J., Gray, J., 1992)

3.1 Types of Parallelism

As pointed out in (Kossmann, D., 1998), the methods for exploiting parallelism in a database environment can be divided into three categories: namely *intra-operator, inter-operator(intra-query), and inter-query parallelism*. In intra-operator parallelism, the major issue is task creation and the objective is to split an operation into tasks in a manner such that the load can be spread evenly across a given number of processors. The second form of parallelism is termed inter-operator parallelism, meaning that several operators within a query can be executed in parallel. This can be achieved either through parallel execution of independent operations or through pipelining Thirdly, parallelism can be achieved by executing multiple queries simultaneously within a multiprocessor system. This is termed inter-query parallelism. (Turek, J., Philip, S., Chan, M.S., Wolf, J.L)

3.2 Exploiting Resources

There are many alternative approaches for the queries to execute at the client machine at which the query was initiated or at the server machines that store the relevant data. These are *query shipping*, which is executing the query at the server side, d***ata shipping,*** which is executing the query at the client, and *hybrid shipping,* which is the combination of above two.

Another important class of system in which queries run over data that is distributed over a number of physical resources is parallel databases. Parallel databases are now a mature technology, and experience shows that parallel query processing techniques are able to provide cost-effective scalability for data-intensive applications. The purpose of a parallel database system is to improve transaction and query response times, and the availability of the system for *centralized* applications.

4 ARCHITECTURE

The proposed design of the system is shown in figure 1. The system implements a multi-threaded runtime environment in order to simultaneously handle queries submitted by multiple users, and also to manage multiple connections with application servers. The system also performs resource balancing between multiple application servers using a replication model that employs statistics that depend on the current and past workload of an application server.

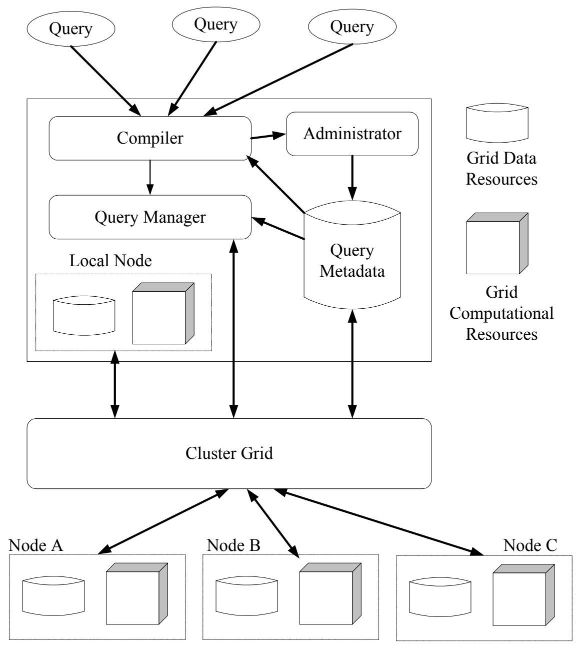

Figure 1: The design of the system

The pseudo-code like representation of the parallel query processing structure is as follows:

```
Input    :  User Query
Output   :  Result of Query
Let  I be Input Query
Let  O be the set of Query Output
Let  R be the result to be output
Let  M be Query Meta-information
Let  T be the Internal Representation of Query
Let  S be the set of sub-Query
Let  $C_i$ be the client to execute the sub-Query $S_i$
Let  P be the Optimal Query plan
Accept I;            // input Query
O  ←    Null;
R  ←    Null;
T  ←    Null;
T  ←    Translate(I, M);  // map user-defined query
        to internal representation
P  ←   OptimizeQuery(T); // generate an
    optimal plan
{S}← SiteSelection(P);  //  Processor Allocation
for $\forall$ $S_i$ in {S} do
{C}       ←  Ship-Query ($S_i$)  //  sent sub-
                          Query to selected node
for $\forall$ $C_i$ in {C}do in parallel
   {O}← ExecuteQuery($S_i$);
        for $\forall$ $O_i$ in {O} do
            R  ←   R $\cup$ ($O_i$);  //  Combine partial
                           results from each node
```

A representative query over employee databases is used as a running example throughout the paper. The query accesses the two employee databases on different data nodes as follows:

```
SELECT  e.name, budget(e.salary)
FROM     Emp e, Dept d
WHERE   e.salary > 100,000 AND
              e.work-id = d.dno;
```

In the query, e and d are two different table views from two databases. Before submitting the query, a global database schema has been constructed to describe and combine the views of the participating databases along with the budget program. The following figure shows the sample evaluation of this query.

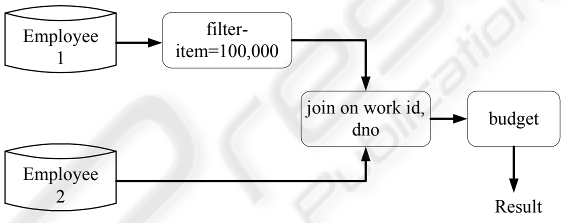

Figure 2: Executing the sample query

We now describe the major components of the system.

MDS (Metadata Directory Service):

The directory service in our system maintains all the information needed in order to parse, rewrite, and optimize a query. It stores the ***schema*** of the database (i.e., definitions of fields, tables, databases, link, integrity constraints, etc.) and the ***partitioning schema*** (i.e., information about what tables have been partitioned and how they can be reconstructed). It also maintains the ***physical information*** such as the location of copies of tables (replicas), information about data nodes and process nodes.

User Query Translation:

The query compiler in our system has responsibility for generating internal representations for Structured Query Language (SQL) which may access data and operation on many nodes. It uses *concept-based approach* to translate the requests of the user query. This approach hides heterogeneity for user. It also takes care of data format, and map from local to global schema. Figure 3 shows the compiler of our system.

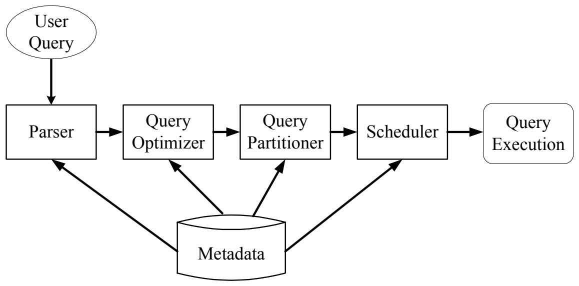

Figure 3: Compiler for Parallel Query Processing

Query Optimization:

Due to advances in network technology, modern distributed systems can become very large and complex. With the advent of automated tools for data analysis and decision support systems, complex queries are becoming common. In order to produce results in minimum response time, query execution needs to be optimized. (Ganguly, S., Hasan, W., Krishnamurthy, R)(Smith, J., Gounaris, A., Watson, P., Norman W., Alvaro A.A., Sakellariou, R., 2003)(Lu, H., Shan, M-C., Tan, K-L., 1991)

The query optimizer in our system transforms internal requirements from translator into a structured query (SQL) depending on the physical state of the system in order to carry out by the execution state. To do this, it follows the two-step optimization paradigm, which is popular for both parallel and distributed database systems. In the first phase, the single node optimizer produces a query plan as if it was to run on one processor. In the second phase, the sequential query plan is divided into several partitions or sub-plans which are allocated machine resources by the scheduler.

One of our objectives of the optimization is to better balance resource utilization. In inter-query parallelism, decisions have to be made on allocating the available processors among a number of competing database operations running in parallel and the overall objective is to minimize the query response time. So, the partitioner and scheduler perform the site selection process based on resource balancing policy. In our system, we propose the siteSelection algorithm is as follows:

```
Input  :  A set of Relations $R_1$ , …, $R_N$
Output: A set of available sites allocated balanced
Let  S be the set of available sites
Let  $S_i$ be the set of sites for Relation i
Let  $site_i$ be the site (resource) for relation i
1:   for i=1 to N do
2:   {  $S_i$  ← Null;
3:       {$S_i$}  ←  Search_Available_Sites($R_i$);
4:       if |$S_i$|  = Null  then
5:  {Si} ←  Find_Minimum_Resource_Site($R_i$);
6:       if |$S_i$|>1 then
7:         {  $site_i$  ←  Select_Best_Site({$S_i$});
8:           {S}  ←  {S} ∪ $site_i$ ;
9:                  update ($site_i$); /* to be located */
10:      }
11:   else
12:      {
13:            {S}  ←  {S} ∪ {$S_i$} ;
14:            update  ($S_i$);  /*  to  be  located  */

15:      }
16:  }
```

The algorithm works one time for each relation. First, it finds the sites that contains relation i. Since there are many copies (replicas) of the table, there will be many sites that match relation i. However, in order to make resource-usage balancing, the method *Search_Available_Sites* in our system finds only the available sites (i.e the site which doesn't use the table i at that time).

If there is no available site i, the method *Find_Minimum_Resource_Site* finds a site that has relation i currently in use but a few other relations taken. In other case, if there are many available sites for i, we further choose the best (i.e most appropriate) table for relation i. Otherwise, we can use the only one site for i. Then we add this site to the set S and update the site to be located. (i.e not available for next relation i)

The basic algorithm for query optimization is Dynamic Programming as follows.

```
Input: SPJ query q on relations $R_1$,…,$R_n$ ,n<= N
Output: A query plan for q
1: for i = 1 to n do {
2:    optPlan({Ri}) = accessPlans(Ri )
3:          prunePlans(optPlan({Ri}))
4:  }
5: for i = 2 to n do {
6:    for all S ⊂ {$R_1$, … , $R_n$} such that |S| = i
         do {
7:            optPlan(S) = null;
8:            for all O ⊂ S do {
9:              optPlan(S) = optPlan(S) ∪
     joinPlans(optPlan(O) , optPlan(S - O))
10:             prunePlans(optPlan(S))
11:            }
12:     }
13: }
14:  finalizePlans(optPlan({$R_1$,….,$R_n$}))
15:  prunePlans(optPlan({$R_1$,….,$R_n$}))
16:  return optPlan({$R_1$,…., $R_n$})
```

The algorithm works in a bottom-up way as follows. First, dynamic programming generates so-called *access plans* for every table involved in the query (Lines 1 to 4 in Figure 8). In the second phase (Lines 5 to 14 in Figure 8), dynamic programming considers all possible ways to join the tables. First, it considers all two-way join plans by using the access plans of the tables as building blocks and calling the *joinPlans* function to build a join plan from these building blocks. In the same way, dynamic programming continues to produce five-way, six-way join plans and so on up to *n*-way join plans. In the third phase (Lines 14 and 15), the *n*-way join plans are managed by the *finalizePlans* function so that they become complete plans for the query.

The figure 9 show the single node logical plan after compilation and the figure 10 shows multi-node physical plan after partitioning and scheduling resources.

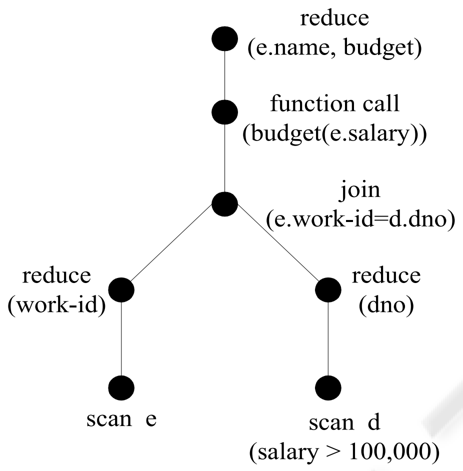

Figure 4: Single node plan

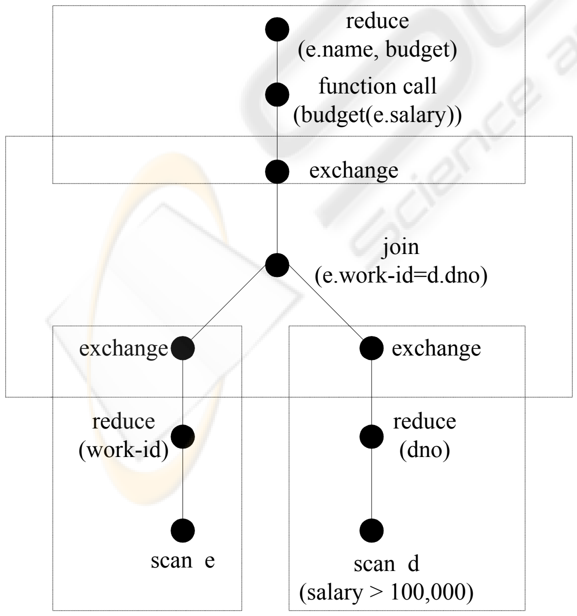

Figure 5: Multiple-node physical plan

Query Execution:

Our system is focused to run on a cluster grid including resources such as PCs, workstations connected by high performance networks/switches (i.e, Gigabytes, Ethernet and Myrinet), running on heterogeneous operating systems (i.e Windows, Linux). In our system, we use ***Multi-Thread Execution*** in which the decomposed and optimized sub-queries are processed as each thread by the servers in parallel. In execution queries, sometime we use ***Data Shipping***, but in many cases, we use ***Query Shipping*** by acting a client as a gateway to make inter-site joins. The final query result is composed by joining the results of these sub-queries by the client. The *Multi-Threaded* model can support a high degree of parallelism. In this model, each thread implements an interface comprising three operations: *send()*, *execute()* and *receive()*. These operations form the glue between the operations of a query plan. The *send* and *receive* calls issued at the MPI level in non-blocking synchronous mode, are managed by multiple user level threads encapsulated in the *exchange* operation. A parallel query can be run as a parallel MPI program over a collection of wide area distributed machines.

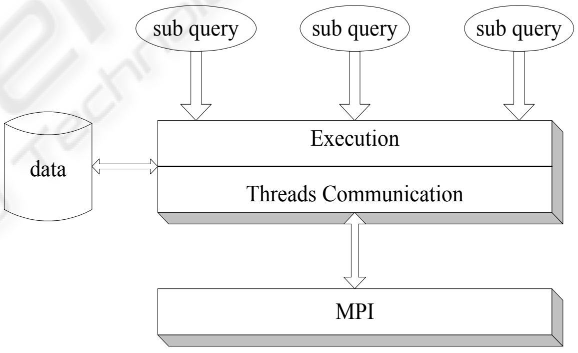

Figure 6: Phases of query execution

The above figure shows the main phases of query execution. The evaluator receives a set of sub queries from the query compiler. First, the sub queries are sent to the Grid nodes through the network and, secondly, they are installed on them via the MPI interface. The next phase involves the execution of the operators comprising the query plan as separated threads. This operator execution may in turn lead to the moving of data between nodes, using a flow-controlled communications infrastructure which itself uses the MPI interface.

5 CONCLUSION

The main goal of this work is to design and implement the framework for supporting data analysis applications in the context of highly distributed environments, such as the computational Grid. Our system concentrates the workload of multiple clients in such a way that it is able to leverage its own computational ability to based on resource balancing capability. This approach results in faster query responses, decreased use of network resources, decreased utilization of possibly remote-located application servers, and effectively better utilization of available resources by partitioning and concurrently executing sub-queries.

REFERENCES

Alpdemir, N., Mukherjee, A., Paton, N. W., Watson, P., Fernandes, A. A. A., Gounaris, A.. and J. Smith., 2003. Service based distributed querying on the grid. In *Proc. of ICSOC*, pages 467–482.

Andrade, H., Kurc, T., Sussman, A., and Saltz, J., 2001. Efficient execution of multiple workloads in data analysis applications. In *Proceedings of the 2001 ACM/IEEE SupercomputingConference*, Denver, CO.

Beynon, M. D., Sussman, A., Catalyurek, U.,. Kurc, T., and Saltz, J., 2001. Performance optimization for data intensive grid applications. In Proceedings of the Third Annual International Workshop on Active Middleware Services (AMS2001), pages 97–105. IEEE Computer Society Press.

Bouganim, L., Florescu, D., Valduriez, P., 1996. Dynamic Load Balancing in Hierarchical Parallel Database Systems. In Proc. of the Int. Conf. on Very Large Data *Bases (VLDB), Mumbai (Bombay), India*.

Dail, H., Sievert, O., Berman, F., Casanova, H., YarKhan, A., Vadhiyar, S., Dongarra, J., Liu, C., Yang, L., Angulo, D., Foster. I., 2003. Scheduling in the grid application development software project. In *Grid resource management: state of the art and future trends*. Kluwer Academic Publishers Group.

DeWitt, D.J., Gray, J., 1992. Parallel Database Systems: The Future of High Performance Database Systems, Communication of the ACM, Volume 35.

Ganguly, S., Hasan, W., Krishnamurthy, R.. "Query Optimization for Parallel Execution", ACM SIGMOD 6/92 California, USA.

Gounaris, A., Norman W., Alvaro A.A., Sakellariou, R. “Resource Scheduling for Parallel Query Processing on Computational Grid”, University of Manachester, Oxford Road, Manchester M139PL, UK