Query Optimization Architecture for Data Grid Environment

Ayouni Houssam Eddine and Belbachir Hafida

Department of Computer Science, Laboratory Systems Signals Data, Faculty of Science
University of Science and Techenology Mohamed Boudiaf-Oran, Algeria

Abstract

Query optimization in data integration systems over large scale network, faces the challenges of dealing with autonomous, heterogeneous and distributed data sources, dynamic execution environment and changing user requirements. In this paper we introduce system architecture for query optimization. The latter consists of several important phases. We introduce also a cost model to calculate the cost of query execution. In order to optimize the cost of query processing considering the constraints in grid environment, an execution model based on mobile agents for efficient execution of binary relational operators (join) of the query is determined based on the cost model. Finally the paper tests our approach through experiment.

Keywords: *data grid, query optimization, database, cost model, mobile agents.*

1. Introduction

Nowadays, the technology of grid Computing allowed the development of services for efficient sharing of computing resources (e.g. CPU, I / O) between multiple sites. A site consists of a large number of users (e.g. scientists, researchers), a very large number of resources (e.g. databases, CPU, memory, program, services) and is often dedicated to an application domain (e.g. pathology, biology, physics). In this context, effective management of resources is fundamental for applications using a grid system.

Resource management plays a very important role in a grid system because it includes the following phases: resource discovery, the selection of resources, the allocation of resources, and the query processing.

The distributed query processing produces a relatively large amount of data transmitted between the different sites participating in the execution of the query. The latter creates a bottleneck in a large scale environment. Therefore, it becomes important to find an execution strategy that reduces the number of transmission and the amount of data transferred during the execution of the query.

The aim of this article is to exploit the problem of query optimization taking into account the characteristics of grid systems (e.g. the large-scale, instability and the autonomy of nodes, heterogeneity and the unavailability of resources). Therefore, our goal is to find a solution that minimizes the cost of execution of queries.

2. Related Work

In last few decades, many researchers have been devoted for the query processing in grid environment [1, 2, 3, 4, 5]. In this context, design and implementation of an efficient query optimization technique for grid environment is utmost important. Taking into account the constraints of the grid, a cost model for calculating the query execution cost, was introduced in [1]. In order to optimize the cost of query processing considering the constraints in grid environment, a linear programming optimization problem (LPP) is formulated based on the cost model., and they also deal a constraint-based query optimization technique using the linear programming optimization problem. In [2], another cost model is defined for dynamic grid database environment, and also gives the dynamic query optimization algorithm used for the query plan to make adaptive evolvement along with the fluctuation of gird environment. The authors of [3] propose a new model for distributed query optimization that integrates three distinct phases namely, (1) creation of single node plan, (2) generation of parallel plan, and (3) optimal site selection for plan execution. They also present different heuristic approaches for solving the proposed integrated distributed query processing problem. In [4], a semantic query optimizer for a grid environment is proposed; it mainly implements optimization of the following three modules: semantic extension of the user query, resources selection, and parallel processing. In [5], the Hameurlain team defined an execution model based on mobile agents to the distributed dynamic query optimization in large-scale systems. The idea is to execute each relational operator using a mobile agent, which allows decentralizing the decisions taken by the optimizer and adapting dynamically to estimation errors on the profile of relations.

In this paper, we propose architecture for query evaluation, which will improve the performance of the query during

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

the execution phase, taking into account the fluctuations of the grid.

3. The proposed approach

In this section, we present our architecture for query processing in data grid environment. We explain the different phases of evaluating a query in our proposed architecture, specifying the role of each phase. We assume that a user submits his query from a node of the local site (transmitter site) via an interface and through authentication mechanisms (predefined by the administrator of the grid). The different phases of the evaluation of the query are:

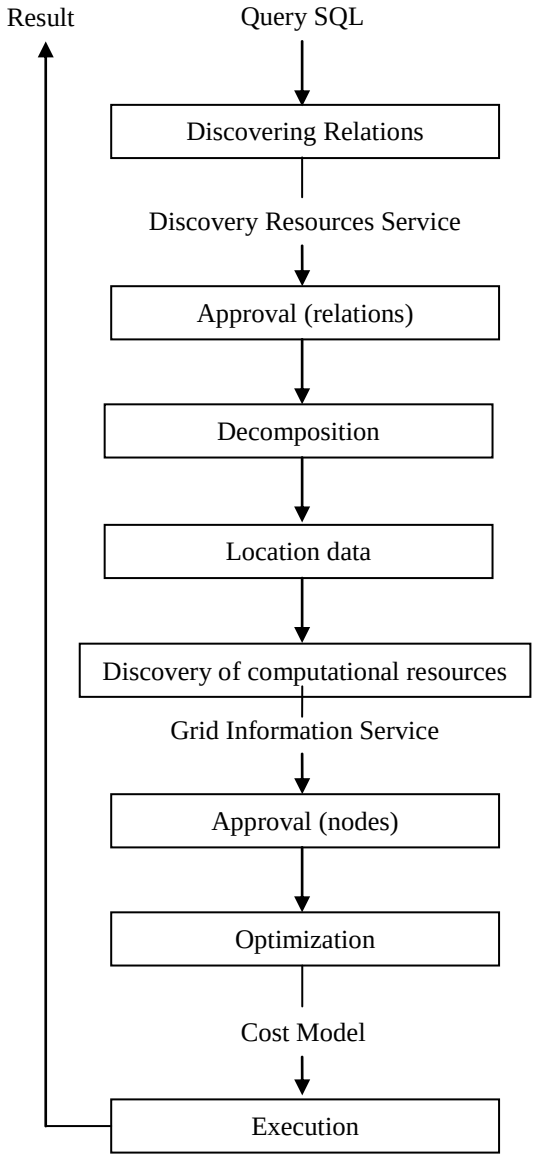


Fig. 1 Architecture of the proposed system.

3.1 Discovering Relations

The first phase is to discover the metadata describing all relations $E = \{E_1, \ldots, E_m\}$ referenced in query. The input parameters of this phase are the names of relations contained in E. Output parameters contain a set of metadata $M = \{MetaDataR_1, \ldots, MetadataR_m\}$ with $MetaDataR_k$ contains metadata describing R_k . Specifically, for a relationR_k, metadata sent are: profile of R_k, and placement of R_k.In profile of R_k , there is the relational schema of R_k(i.e. attributes name and type of R_k) and statistical values of R_k (e.g. the minimum value of an attribute and the number of distinct values).These statistical values are used during the optimization phase. Regarding the placement of R_k, it contains information relating to the location, the fragmentation and duplication of R_k.

3.2 Approval (Relations)

It is important to conduct a phase of approval in an unstable system. The approval means that checks whether at least (in the case where a relation is replicated on multiple nodes) one node storing a relation (or fragment of a relation) is connected to the system and it is possible to reach this node. Indeed, it would be better to stop the evaluation of the application at this stage and return a failure message to the user if a node storing a relation or fragment is not available.

3.3 Decomposition of queries

The entry of this phase is a distributed query SQL. As in systems of distributed databases [6], the query decomposition undergoes four stages: (i) normalization, (ii) analysis, (iii) removing redundancy and (iv) the rewriting. Steps (i), (iii) and (iv) based on a set of transformation rules of query to perform better. By cons, step (ii) is based on part of the set *M*. This is the part about the profile of relations referenced in query, specifically the attributes of these relations. Indeed, the analysis step is responsible for performing the syntactic and semantic analysis. The syntactic analysis can check whether the name of an attribute of relation referenced in query indeed on the relational schema. As for the semantic analysis, it can check, for example, if the relationship Doctor means a medical or doctor of science. The output of this phase is an "good" algebraic query in the sense that incorrect requests are rejected, without redundancy, expressed in global relations.

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

3.4 Location Data

The input of this phase is a query expressed on distributed algebraic relations. The location data is performed based on some of the metadata contained in the set *M*. Specifically, it is metadata about the placement of relations. As in systems of distributed databases [6], the location of data consists in transformation of distributed query (expressed in global relations) on a distributed query equivalent expressed on fragments. Specifically, the location of a distributed query involves two steps [7]: (i) generating an equivalent canonical query and (ii) simplification. The canonical query (i) is generated from the distributed query expressed on overall relations by replacing each relation with its corresponding query reconstruction. The reconstruction of a relation from its fragments is performed by a relational operation (e.g. a join). Simplification (ii) consists to remove unnecessary operations that can, for example, produce an empty result.

3.5 Discovery of Computational Resources

This phase consists to find only static metadata describing the computational resource [8]. It's consists to discovery a sub-set of nodes available for the execution of a query. In this context, a simple strategy is to discover computing resources (e.g. CPU or memory of a site). The discovery strategy is based on the Grid Information Service (GIS) [9], it is a directory that contains static metadata describing the sites.

3.6 Approval (nodes)

Along with the approval relations phase, it is important to conduct a phase of approval for computing resources to see if they are well connected to the system. This phase aims to eliminate discoveries computing resources not available due to an event such as a network outage.

3.7 Optimization

The goal of query optimization is to determine a strategy for query execution that minimizes a cost function. The optimization phase includes: (i) selecting the best replication of relations, (ii) selecting the best nodes among the set of nodes discovered, (iii) the definition of the order of relational operations and the best access algorithm for each of them, and (iv) the placement of relational operators (also called resource allocation) for the execution of operations. The optimizer is based on a cost model consisting essentially of (i) a set of metrics (e.g. response time of a join) and (ii) a library. The metric (i) are calculated based on parameter hosts values (i.e. value of the CPU load, load value of I/O and value of the memory load of a node) and network (i.e. the value of the latency and bandwidth between two nodes) stored in the library. The library (ii) contains statistics on the system and relations, and cost formulas for estimating the cost of query execution. Several approaches are proposed for estimating the cost of executing a query (e.g. the calibration approach [10, 11], the sampling approach [12] or also in the historical approach [13]). The main problem in most of these approaches is that they do not take into account the variation values of parameter host and network during the execution cost estimation of a relational operation (e.g. a join). This may have limits in a system where resources are shared on a global scale and where each user (e.g. an application) can connect to a remote node to perform tasks (e.g. submission of a query). To take into account the variation of parameter values hosts and networks, we propose the use of a model of execution based on mobile agents which can improve the accuracy of the metrics provided by the cost model.

3.8 Execution

In the last phase, we resume the work of [14, 15, 16] that defines an execution model based on mobile agents for dynamic optimization of distributed query in large-scale environments. A mobile agent is an autonomous and adaptable software entity, able to move dynamically (code, data), to access data or resources from a remote location. This extension allows them to change their execution sites proactively. Each mobile agent executing a join chooses itself its execution site by adapting to the execution environment (e.g. CPU load, bandwidth) and the estimation accuracies on temporary relation sizes. Hence, the control which makes the decision of the execution site change is carried out in a decentralized and autonomous way.

4. Cost Model

In this section, we describe the different parts of the cost model involved in the estimation phase of the cost of query execution plans. We distinguish two parts: the resident part on the site, and integrated part in the mobile agent. This cost model is used to estimate the cost of execution plans of queries submitted.

4.1 Site Cost Model

The cost model of a site ***Si*** consists of four units: (i) the profile of each data source known by ***Si***, (ii) the characteristics of ***Si*** (host parameters + network parameters), (iii) the characteristics of other sites that interact with ***Si***, and (iv) the cost formulas.
This information is provided by the GIS, and stored in the data catalog with a time-stamp of the last update date, except the cost formulas, because they will not need

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

frequent updating. The time-stamp allows using the most recent data.

4.2 Agent Cost Model

The agent cost model is supplied to the agent during the query optimization. The agent cost model is composed of two units: (i) the migration space (i.e. list of sites has the same data source), (ii) location of the second mobile agent (remote mobile agent that deals with the second data source). A mobile agent can consult the site cost model, to estimate its parameters (production cost, and the migration cost).

4.3 Cost Formulas

In a grid environment, the resources that computing task needed are computing resources such as memory, CPU etc. Hence, computing task can be assigned to any node that has enough computing resources for that task.

4.3.1 Nodes Cost Formulas

The operators of a query execution plan will be executed at the nodes of the sites. Due to the autonomy and heterogeneity of nodes (data sources), it becomes difficult to determine the cost of processing an operator on the latter. To address this insufficiency, we used a simulation model presented in [16]. This allows, among other things, to estimate for a given query, the response time and the volume of results. To obtain these values, statistics on the data manipulated and formulas cost associated with each operation are used (e.g. join, selection). This model contains calibrated information concerning the execution cost of every operator [17], and generic cost formulas. This information represents the initialization processing cost, the processing cost of a tuple and the production cost of the result [17]:

- SSO initial cost for sequential scan, estimated based on the disk load, calculated by the GIS,
- SS1 cost for fetching a tuple, estimated based on the CPU load of node, calculated by the GIS,
- SS2 cost of processing a result tuple for sequential scan, estimated based on the memory load of node, calculated by the GIS,
- FS(p) represents the selectivity factor of the predicate of the query which can be estimated by selectivity formulas described in [18].

This information is used to estimate the response time of operators executed on nodes, using generic formulas. For example, the formula estimating the cost of scan of a relation R not indexed is described in the following way [18]:

$$Scan\_Cost\ (R) = SS0 + SS1 * ||R|| + SS2 * ||R|| * FS\ (p) \quad (1)$$

In the case where multiple nodes connected to the same site, has given the same data R, the node check *min(Scan_Cost (R))* will be selected.

4.3.2 Sites Cost Formulas

These formulas are used to estimate the response time of operators executed on the Site. They are more accurate than those the nodes, because the architecture of the site and the algorithms which implement the operators are known. In our case, we are interested only binary operators such as joins, because unary operators (projection, selection) run locally, and therefore only the cost of processing to be calculated, it is usually negligible compared to the cost of communication in large scale systems such as data grids.
We have extended the semi-join algorithm by allowing the agent to choose the site of execution of the join in a data grid. The formula for estimating the response time of the semi join based on mobile agents with two basic relations T = Join (R, S) performed by two mobile agents AG1 and AG2, is as follows:

$$Cout\text{-}Semi\text{-}Join_{R,S} = \text{Projection-Cost(R)} + \text{Join-Cost(S, temp1)} + Join\text{-}Cost(R, temp2) + \sum Migration\text{-}Cost(AG1) + \sum Migration\text{-}Cost(AG2) + \text{CoutTrans}_{S1,S2}(\text{temp1}) + \text{CoutTrans}_{S2,S1}(\text{temp2}) \quad (2)$$

with :

$$Projection\text{-}Cost(R) = SS0 + SS1 * CARD(R) + SS2 * CARD(R) \quad (3)$$

$$Join\text{-}Cost(R, S) = Scan\_Cos(R) + CARD(R{:}P) * FS(p) * Scan\_Cost\ (S) \quad (4)$$

with

$$FS(p) = 1.5/maw(CARD(R), CARD(S))$$

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

$$CoutMigration_{Si,Sj}(AG) = \begin{cases} 0 \quad \textbf{If AG does not move} \\ CoutSer + CoutDeser \quad \textbf{If AG moves without relation from } S_i \textbf{ to } S_j \\ CoutSer + CoutDeser + CoutTrans_{Si,Sj}(Rel) \quad \textbf{If AG moves with relation from } S_i \textbf{ to } S_j \end{cases} \quad (5)$$

$$CoutTrans_{Si,Sj}(Rel) = Initial(Si,Sj) + \left(\frac{/Rel/}{/p/}\right) * Trans(Si,Sj) \quad (6)$$

The terms used in the above formulas have the following meanings:

- *Projection-Cost(R)* : The projection cost of R,
- *Join-Cost(R, S)*: The natural join cost between R and S,
- $CARD(R)$: The number of rows from R.
- *CARD(R:P):*The number of rows from R that satisfy the join condition P.
- $Migration - Cost_{Si,Sj}(AG)$: The sum of the AG migration cost. The migration cost of an agent [19] is the sum of the serialization cost ($CoutSer$), the de-serialization cost of the context of the agent ($CoutDeser$) and the transfer cost (*CoutTrans*),
- $Initial(Si,Sj)$: Time to establish communication between two sites Si and Sj estimated based on the value of the latency between Si and Sj,
- $CoutTrans_{Si,Sj}(Rel)$: The transfer cost of the relation T depends mainly on the localization of the agent and networks characteristics,
- */Rel/*: The size of Rel.
- */p/*: The size of a page p.

In this context, we conclude the total cost of the query, which is the sum of the costs of the query operators:

$$Cost_{query} = \sum Cout\text{-}Semi\text{-}Join_{R,S} \quad (7)$$

5. Performance Evaluation

In this section, the performance of the proposed system is studied. The first part describes the general configuration of the simulation, including simulated grid environment. The second part shows the results of the experiment and gives a brief analysis.

5.1 Simulation Tool

We have implemented our proposed system using the GridSim toolkit. The GridSim toolkit [20] is one of the most widely used grid simulation tools.
It has been used for simulating and evaluating virtual organizations based resource allocation, workflow scheduling, and dynamic resource provisioning techniques in global grids.

5.2 GridSim Configuration Files

There are four configuration files used to control various inputs to GridSim. These are as follows:

5.2.1 Simulation Parameter File

It contains various simulation parameters which the user can modify them. These parameters are the names of configuration files of the network, files, resources, and users of the grid.

5.2.2 Network Configuration File

It describes the network topology, that is to say the available routers, and network connections between them. The grid configuration that we have used in our simulation is the CMS Data Challenge 2002 test bed [21] (Figure 2).

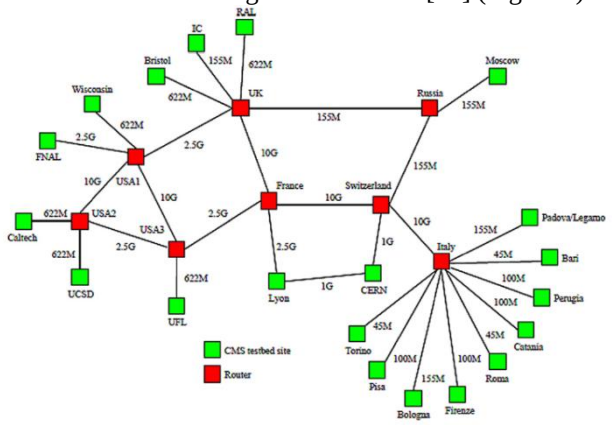


Fig. 2 CMS Data Challenge 2002 Grid Topology.

5.2.3 Files Configuration File

It contains information on the simulated files like size of each file and its identifier. Was used in our simulation "Employees Sample Database" located on the official website of MySQL, and contains the following relations (Table 1):

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

Table 1: Employees Simple Database.

Table Name	Number of tuples	Size (mégaoctet)
employees	300024	14.5156
departments	9	0.0313
dept_manager	24	0.0469
dept_emp	331603	21.5469
titles	443308	30.1094
salaries	2844047	130.1875

5.2.4 Resources Configuration File

It contains the description of Grid resources such as name, size of storage resources, bandwidth, router name on which the resource is attached, list of files (replicas) containing in each resource.

5.2.5 Users Configuration File

It contains descriptions of grid users such as name, name of the router on which the user is connected, the catalog replicas name (in our case we use a global catalog replicas "TopRegionalRC."), and the list of operations performed for each user.

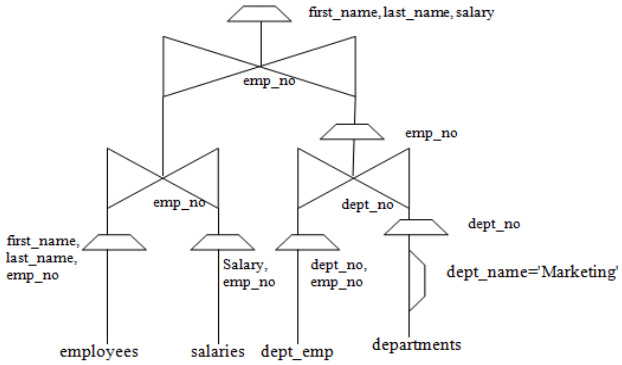
5.3 Simulation Results

To test the validity and performance of our approach, two different optimization techniques: the classical approach of query optimization for distributed databases (DQO) [22], and an algorithm for query optimization in data grid environments (GQO), described in [1], are compared with our approach to process a query (figure 3) in a simulated grid. The following code is an example of query which integrates four tables in the database "Employees Sample Database."

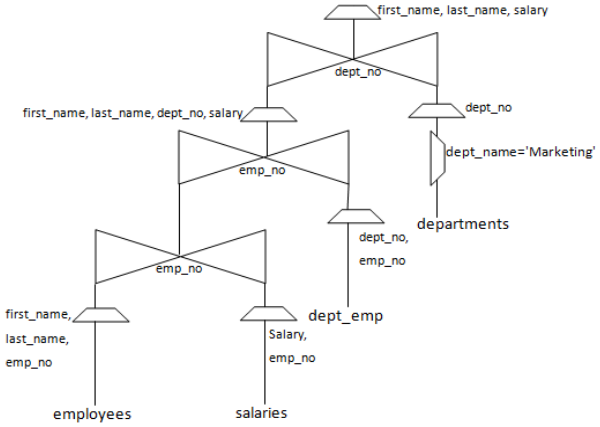
```
select first_name, last_name, salary
from employees,salaries,dept_emp, departments
where
(employees.emp_no=salaries.emp_no)and(employe
es.emp_no=dept_emp.emp_no)                    and
(dept_emp.dept_no=departments.dept_no)and(dep
artments.dept_name='Marketing');
```

Fig. 3 Example of query.

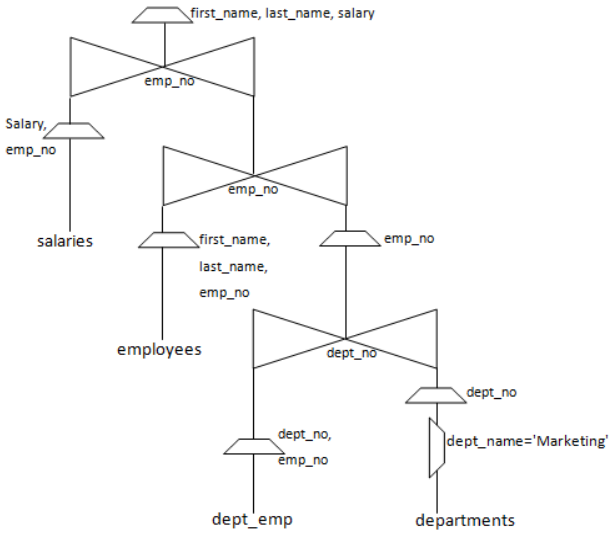
After applying the rules of the decomposition, and eliminate redundancies by applying the properties of the relational algebra, three execution plans of query are generated (Figure 4).



(a) Bushy Tree



(b) Deep left tree



(c) Deep right tree

Fig. 4 Execution Plans of query.

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

Table 2: The simulation parameters.

Number of Sites	10
ss0	$10 \times 10^{-6}\ \mu s$
ss1	$30 \times 10^{-6} \mu s$
ss2	$20 \times 10^{-6} \mu s$
Page size	4 Ko
Cost of serialization	3.2 ms
Cost of deserialization	4.3 ms
Agent size	2806 Bit
The minimum reliability of the bandwidth between sites	50%
The maximum reliability of the bandwidth between sites	90%
Number of experiment	100

The query is applied with the three techniques of query optimization in the same environment, and the final results are shown in Figure 5 for the three execution plans.

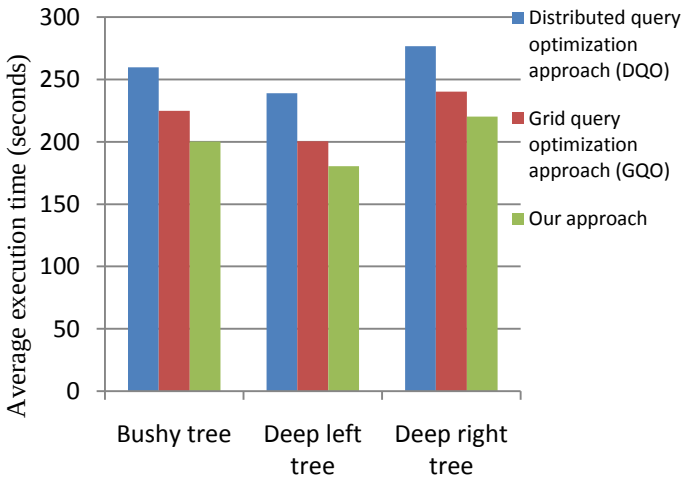


Fig. 5 Evaluation of the average execution time of different approaches for each execution plans.

The results in Figure 5 show that our approach handles the query in a better delay compared to other approaches DQO and GQO in all the cases.
Based on this analysis, we constant that the performance of our approach is better, the latter is due to its ability to adapt with the changing environment (load nodes and bandwidth) based on power mobile agents.

6. Conclusion

In this paper is presented architecture of a query optimization in a grid environment. Our architecture can handle a request in the shortest time. In the optimization technique, model of mobile agents are used in order to deals a problems of the dynamic behavior and heterogeneity of the grid resources. A cost model for estimating the cost of query is also introduced.
The results obtained are very satisfactory and makes our approach a tool capable to process queries in a grid environment with an optimal execution time. The use of mobile agents increases the performance of our approach because they adapt well to changes in the dynamic environment that characterizes the data grids, with migrating through the nodes to improve overall response time of the query.

References

[1]Chhanda Ray, Nilava Guha ”Determination of Cost Model for Constraint-based Query Optimization in Data Grids “Proceeding of International Conference on Advances in Computer Science ACEEE 2010.
[2] N. Hu, Y. Luo, Y. Wang, “Adaptive Evolvement of Query Plan based on low cost in Dynamic Grid Database”, proceedings of 9th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pp 411-415, 2008.
[3] Krishnamoorthy, S., Saple, A.K., and Achutharao, P.H. An integrated query optimization system for data grids. In Proceedings of Bangalore Compute Conference 2008.
[4] Luo Y H, Chen T F, Zhang Y S. Study on semantic query optimization of grid data. Computer Engineering and Applications, 45(2):16-20,2009.
[5] Franck Morvan, Abdelkader Hameurlain. A Mobile Relational Algebra. in : International Journal of Mobile Information Systems, IOS Press, Vol. 7 N. 1, p. 1-19, january 2011.
[6] M. T. Özsu, P. Valduriez; Principles of Distributed Database Systems, Second Edition Prentice-Hall, 1999.
[7] G. Gardarin and P. Valduriez ; SGBD Avancés, Bases de données objets, déductives et réparties, Eyrolles, 1990.
[8] M. El Samad, A. Hameurlain, F. Morvan; Resource Discovery and Selection for Large Scale Query Optimization in a Grid Environment, International Conference on Systems, Computing Sciences and Software Engineering (SCSS 2007), Springer, Advances in Computer and Information Sciences and Engineering, pp. 279-283, 2007.
[9] B. Plale, P. Dinda, M. Helm, G. von Laszewski, and J. McGee. Key concepts and services of a grid information service. 2002.
[10] W. Du, M.-C. Shan ; Query Processing in Peagasus. In Object-Oriented Multidatabase systems : A Solution for Advanced Applications, pp. 449-468, 1995.
[11] G. Gardarin, F. Sha, Z.-H. Tang ; Calibrating the Query Optimizer Cost Model of IRODB, an Object-Oriented Federated Database System, Proc. of 22th International Conference on Very Large Data Bases, Morgan Kaufmann, Mumbai (Bombay), India, pp. 378-389, september 1996.
[12] Q. Zhu, S. Motheramgari, Y. Sun: Cost Estimation for Queries Experiencing Multiple Contention States in Dynamic Multidatabase Environments. Knowl. Inf Syst.5(1): 26-49,2003.

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

[13]S. Adali et al. ; Query Caching and Optimization in Distributed Mediator Systems, Proc. of ACM SIGMOD International Conference on Management of Data, ACM Press,Montreal, Canada, pp. 137-148, june1996.
[14] J.-P. Arcangeli et al.; Mobile Agent Based Self-Adaptive Join for Wide-Area Distributed Query Processing, Journal of Database Management, Idea Group, Vol. 15, N. 4, pp. 25-44, 2004.
[15] M. Hussein et al.; Embedded Cost Model in Mobile Agents for Large Scale Query Optimization, Proc. of the 4th Intl. Symposium on Parallel and Distributed Computing, IEEE CS, pp. 199-206, 2005.
[16] F. Morvan, M. Hussein, A. Hameurlain; Mobile Agent Cooperation Methods for Large Scale Distributed Dynamic Query Optimization. Dans : International Conference on Database and Expert Systems Applications (DEXA 2003), IEEE Computer Society, pp.542-547, september 2003.
[17] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding Code Mobility. IEEE Trans. on Software Engineering, May 1998.
[18] W. Du, M. Shan. Query processing in Pegasus. In Object-Oriented Multidatabase Systems, A Solution For Advanced Applications, Prentice Hall International (UK) Ltd, pp. 449-468, 1995.
[19] L. Ismail, D. Hagimont: A Performance Evaluation of the Mobile Agent Paradigm. OOPSLA 1999: 306-313.
[20] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," The Journal of Concurrency and Computation: Practice and Experience, Vol. 14, Issue 13–1, Wiley Press (2002).
[21] K. Holtman, CMS data grid system overview and requirement, Tech. Report CERN July, 2001.
[22] Evrendilek, C., Dogac, A., Nural, S., Ozcan, F.: Multidatabase Query Optimization. Journal of Distributed and Parallel Databases 5(1), 77–113 (1997).

Ayouni Houssam Eddine received her Master. degree from the Department of Computer Science at the University of Science and Technology Mohamed Boudiaf USTO-Oran, Algeria, in 2009. Currently he is a phd student at the University of Science and Technology Mohamed Boudiaf USTO. His research areas include Grid Computing, Query Optimization, and Mobile Agents.

Belbachir Hafida leads a many research projects on Database, Data mining, and Grid Computing. She is the manager of Laboratory Systems Signals Data, Department of Computer Science, Faculty of Science.

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.