International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:6, 2007

Database Placement on Large-Scale Systems

Cherif Haddad and Faouzi Ben Charrada

Abstract**—**Large-scale systems such as Grids offer infrastructures for both data distribution and parallel processing. The use of Grid infrastructures is a more recent issue that is already impacting the Distributed Database Management System industry. In DBMS, distributed query processing has emerged as a fundamental technique for ensuring high performance in distributed databases. Database placement is particularly important in large-scale systems because it reduces communication costs and improves resource usage. In this paper, we propose a dynamic database placement policy that depends on query patterns and Grid sites capabilities. We evaluate the performance of the proposed database placement policy using simulations. The obtained results show that dynamic database placement can significantly improve the performance of distributed query processing.

Keywords**—**Large-scale systems, Grid environment, Distributed Databases, Distributed query processing, Database placement

I. Introduction

DISTRIBUTED query processing has attracted a lot of research attention in the last two decades. These efforts essentially concentrate on proposing strategies and algorithms to minimize response time while minimizing resource consumption [1]. In new Internet-based environments, like Grids, such strategies have to be re-evaluated to fit for the large-scale context [2]. In this new context, conventional query processing strategies with the homogeneous assumption on resources will not work well, because they are unable to adapt to unexpected changes in the performance of the communication networks and computing resources.

In large-scale systems, we have to answer the fundamental question traditionally addressed by the database community: "How to efficiently manage and query large volumes of widely distributed data?" The problem is still the same, but the situations become more complex: data access over wide-area networks involves a large number of remote data sources, intermediate sites and communication links, all of which are vulnerable to congestion and failures [3]. Query processing is now placed in the context of wide-area and dynamic environments instead of local and static environments. Thus, it becomes problematic due to the changes coming from both underlying system and user requirements [4].

In Grid context, a user submits database queries from his workstation, which is located at a particular site on the Grid, and requires that the queries be executed as fast as possible. To execute a query, three kinds of resources are needed: computational resources, data resources and network resources. Ideally, the query processing should be able to optimize the usage of these three kinds of resources. Optimization should be carried out based on the status of Grid resources (workload of computing elements, location of data, network load).

In distributed query processing, query operators can be placed at sites in a way that minimizes expected communication costs, execution time or other metrics [5]. These decisions are based in large part on knowledge of which data is located at which sites. This *circular dependency* between data placement and query optimization has significant performance implications for distributed database systems. The challenge then is to integrate a database placement policy and the query optimization in an efficient and effective manner in dynamic and large-scale systems.

In this paper, we study the technique for combining database placement and query optimization in Grid environments. We propose a dynamic database placement policy and we describe how it can be integrated into a query optimizer.

The remainder of this paper is organized as follows. Section 2 discusses the new challenges that arise when we integrate database placement policies with query optimization in large-scale and dynamic systems. Section 3 overviews briefly the Grid environment. Section 4 presents our database placement policy. Section 5 gives and discusses our experiment results. Section 6 concludes the paper and gives some suggestions for future works.

II. Database Placement and Query Optimization

A query optimizer decides which methods to use to execute query operations, in which order, and at which site [6]. To make these decisions, the optimizer enumerates alternative plans and chooses the best one using a cost estimation model [1]. Indeed, the query optimizer have to answer the following question: "Given a query and the current location of data and other parameters, how can this query be executed in the cheapest or the faster possible way?"

The database placement addresses the problem of determining where to place a fragmented database on a network of storage elements. Database placement policies are important because appropriate placement of database fragments reduces bandwidth consumption and improves response time [7]. While database placement is important, a novel aspect of database placement is his integration with a query optimizer. Rather than having a distinct process to perform the database placement, it is preferable that database

Manuscript received January 16, 2006.
Ch. Haddad is with the Department of Computer Science, Faculty of Sciences of Tunis, 1060 Tunis, Tunisia (corresponding author phone: +21698830513; e-mail: cherif.haddad@gmail.com).
F. B. Charrada is with the Department of Computer Science, Faculty of Sciences of Tunis, 1060 Tunis, Tunisia (e-mail: f.charrada@gnet.tn).

placement leads the query optimizer to generate query plans that result in fragment re-allocation or fragment replication in order to enhance the performance of a DBMS. Database placement can be implemented as a module outside of the query optimizer. This module influences the optimizer to sometimes make operator site selection decisions for some queries in order to perform a database placement that will be beneficial for later queries. As a result, we integrate database placement with a query optimizer as shown in Fig. 1.

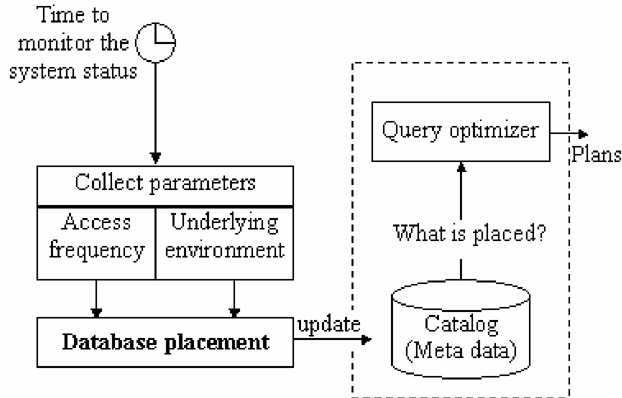

Fig. 1 Integrating database placement with query optimizer

When the optimizer requests the current data locations, the database placement policy enhances the answer with new fragment locations that can be proposed. Note that the database placement do not changes the optimizer's search strategy and cost model.

In a large-scale and dynamic environment such as the Grid, the database placement must also considers changes coming from underlying environment (e.g., resources stability, network bandwidth and workload). A number of Grid characteristics distinguish the Grid database placement problem from database placement strategies suggested in traditional distributed environments [8]:

- Heterogeneity of resources;
- Multiple administrative domains;
- Large-scale;
- Dynamic characteristics of resources.

III. Grid Environment

We suppose a Grid environment in which users submit database queries from any one of a large number of sites. We adopt the same view of a data grid as that proposed by the European Data Grid [7]. The main resources of a Data Grid are:

- The computing elements provide the Grid users with CPU cycle for query execution. Each computing element is located at a particular site on the Grid.
- The storage elements provide the Grid users with storage capacity. Each computing element is located at a particular site on the Grid.
- The Network provides Grid users with bandwidth for data transfer.

We model a Grid as a set of sites, each comprising a number of computing and storage elements, a set of users and a set of database fragments. Each site can have a different number of computing and storage elements. Sites are connected together by WAN's limited bandwidth and computing elements within a site are joined together over a local area network.

IV. Database Placement Policy

This section presents our proposed database placement policy. First, we enumerate the placement parameters. Second, we present our database placement algorithms. Finally, we discuss what and when the database placement policy is triggered.

A. Placement parameters

The parameters considered in our database placement policy are:

1. *Site parameters*: Each Grid site is denoted by GS_a and for each site GS_a, $LANBW(GS_a)$ represents the LAN bandwidth of GS_a;
2. *Storage element parameters*: Each storage element is denoted by SE_i and for each storage element SE_i, $Site(SE_i)$ represents the site where SE_i is located, $SR(SE_i)$ represents the space reserved to store database fragments, $STAB(SE_i)$ represents the stability of SE_i which encompasses storage element failures, communication failures and the disconnection of the storage element from the grid, $DISKBW(SE_i)$ represents the disk bandwidth of SE_i;
3. *Network parameters*: The communication cost $C(GS_a,GS_b)$ between two sites GS_a and GS_b represents the average delay of sending one unit of data (1KB) from one site to another;
4. *Database parameters*: We consider a relational database DB as a collection of m replicated fragments $\{F_1,F_2,...,F_m\}$. The database is vertically partitioned into fragments (attribute-based partitioning). For each fragment F_k, $Size(F_k)$ represents the size of fragment F_k. F_k^l represents the replica l of fragment F_k. $SE(F_k^l)$ defines the identity of the storage element where the replica F_k^l is actually located;
5. *Fragment parameters*: For each fragment F_k, let r_{GS_a,F_k} be the number of F_k replicas in site GS_a and r_{F_k} be the total number of F_k replicas in the Grid;
6. *Query type*: Let $Q = Q_R \cup Q_W$ be a set of n queries, $Q=\{q_1,...,q_n\}$ where Q_R is the set of read queries and Q_W is the set of write queries. We use the following notations and assumptions:
   - For each query q_r, $Site(q_r)$ represents the site where q_r is submitted;
   - $Freq(q_r)$ defines the frequency execution of query q_r;
   - Let Q^{Fk} be the set of queries which access to fragment F_k;
   - The selectivity parameter, $Sel(q_r, F_k)$, is defined as the percentage of F_k to be accessed by q_r;
   - Let $Upd(q_r, F_k)$ be the size of message sent by q_r to

update fragment F_k.

B. Database placement algorithms

To reduce the access latency for the query processing, it is beneficial to place the fragments accessed in a query close to each other on the same site. Also, some frequently accessed fragments must be replicated to reduce the access time. In this subsection, we propose two algorithms to dynamically re-allocate and to replicate a number of database fragments while meeting the query access patterns and Grid sites capabilities. Since we deal with a high-scale environment, the database placement algorithms are just responsible for its own site. Each site uses a set of tools to obtain information about the state of the system [9] and takes database placement decisions.

1) Fragment replication algorithm

The main task of a database placement policy is to determine which fragment must be replicated at the query site. A fully replicated database is not optimal since the update propagation to each fragment replica takes overtime and hence decreases the response time for write queries. However, partially replicated database do not provide optimal read response times since only parts of the database fragments are replicated. So, the access frequency of read and write queries has to be taken into account. We suppose that the communication over the Grid WAN network is very expensive. Thus, we try to perform a database fragment replication due to their individual access frequency. The decision of whether or not the fragment should be replicated in a Grid site depends on:

(i) the fragment read cost $ReadCost(F_k)$ defined as:

$$ReadCost(F_k) = \sum_{r=1}^{n} C(Site(q_r), Site(F_k)).Sel(q_r, F_k).Size(F_k).Freq(q_r)$$

Where $q_r \in Q_R \cap Q^{Fk}$

(ii) the fragment write cost $WriteCost(F_k)$, defined as:

$$WriteCost(F_k) = \sum_{r=1}^{n} \sum_{l=1}^{rFk} C(Site(q_r), Site(F_k)).Upd(q_r, F_k).Freq(q_r)$$

Where $q_r \in Q_W \cap Q^{Fk}$

(iii) the fragment external read cost $AC^{Ext}_{GS_a,F_k}$, defined as:

$$AC^{Ext}_{GS_a,F_k} = \sum_{r=1}^{n} C(Site(q_r), Site(F_k)).Sel(q_r, F_k).Size(F_k).Freq(q_r)$$

Where $q_r \in Q_R \cap Q^{Fk}$, $Site(q_r) = GS_a$ and $Site(F_k) \neq GS_a$

Clearly, if $AC^{Ext}_{GS_a,F_k} > 0$, many queries have to be served from other sites. This creates higher WAN network communication costs. To minimize the communication cost generated by read queries we would try to reduce the cost of external read accesses. A way to reduce the cost of external read accesses is to create and store a F_k replica in site GS_a. A database fragment F_k is considered as candidate to be replicated in a site GS_a if: (i) $r_{GS_a,F_k} = 0$; (ii) $ReadCost(F_k) >> WriteCost(F_k)$; and (iii) $AC^{Ext}_{GS_a,F_k} > 0$.

The fragment replication algorithm takes as input a Grid site GS_a and a set of candidate fragments $\{F_k\}$ to be replicated and placed in the site and returns a placement P. Due to the fact that stability of storage elements $STAB(SE_i)$ can vary dynamically, storage elements with a high stability are advantaged. If there are many storage elements with the same stability, we choose storage elements that have a large disk bandwidth and a large available space disk. The fragment replication algorithm is given by algorithm 1.

Algorithm 1 FRAGMENT REPLICATION ALGORITHM

Require: GS_a: Grid site,
$\{F_k\}$: Set of fragments to be replicated

Ensure: Placement P

1. $P = \varnothing$
2. Let $F = \{F_k\}$ /* set of fragments sorted in a decreasing order of $Size(F_k)$ */
3. Generate a set $SE_{GS_a} = \{SE_i\}$ of candidate storage elements of site GS_a sorted in a decreasing order of $\langle STAB(SE_i), DISKBW(SE_i), SR(SE_i)\rangle$
4. **For all** F_k in F **do**
5. $SE_i = GetFirstItem(SE_{GS_a})$
6. **While** (F_k is not placed) **do**
7. **If** $SR(SE_i) >= Size(F_k)$ **then**
8. $P = P \cup \{(F_k, SE_i)\}$
9. $SR(SE_i) = SR(SE_i) - Size(F_k)$
10. **End If**
11. $SE_i = GetNextItem(SE_{GSa})$
12. **End While**
13. **End For**

Output: P

The time complexity of the fragment replication algorithm is $O(nm)$, where n is the average number of storage elements per site and m is the number of fragments to be replicated.

2) Fragment re-allocation algorithm

The Grid environment keeps changing all time. Sites which contain database fragment replicas can leave the Grid and new sites with computing and storage capabilities can be added. Due to this specific characteristic, the database fragments must be re-allocated to ensure efficient resource usage. The decision of whether or not the fragment should be re-allocated in a Grid site depends on the stability of its storage element and the storage capabilities of the new storage elements added to the site. First, we generate a set $SE_{GS_a} = \{SE_i\}$ of candidate storage elements SE_i in a decreasing order of $\langle STAB(SE_i), DISKBW(SE_i), SR(SE_i)\rangle$. Then, the set of candidate fragments to be re-allocated is defined as: $\{F_k\}$ where $SE(F_k) \notin SE_{GS_a}$.

Fragment re-allocation algorithm gets as input (GS_a,$\{F_k\}$, P) describing, resp., the Grid site, a set of fragments to be re-allocated and the current placement and returns a new placement P^{new}. The fragment re-allocation algorithm is given by algorithm 2.

Algorithm 2 FRAGMENT RE-ALLOCATION ALGORITHM

Require: GS_a: Grid site, $\{F_k\}$: Set of fragments to be re-allocated, P: Current placement
Ensure: New Placement P^{new}

1. $P^{new} = P$
2. Let $F = \{F_k\}$ /* set of candidate fragments sorted in a decreasing order of $Size(F_k)$ */
3. Generate a set $SE_{GS_a} = \{SE_i\}$ of candidate storage elements of site GS_a sorted in a decreasing order of $\langle STAB(SE_i), DISKBW(SE_i), SR(SE_i)\rangle$
4. **For all** F_k in F **do**
5. $SE_i = GetFirstItem(SE_{GS_a})$
6. **While** (F_k is not placed) **do**
7. **If** $SR(SE_i) >= Size(F_k)$ **then**
8. $P^{new} = P^{new} - \{(F_k, SE(F_k))\}$
9. $P^{new} = P^{new} \cup \{(F_k, SE_i)\}$
10. $SR(SE_i) = SR(SE_i) - Size(F_k)$
11. $SR(SE(F_k)) = SR(SE(F_k)) + Size(F_k)$
12. **End If**
13. $SE_i = GetNextItem(SE_{GSa})$
14. **End While**
15. **End For**

Output: P^{new}

The time complexity of the fragment re-allocation algorithm is $O(nm)$, where n is the average number of storage elements per site and m is the number of fragments to be re-allocated.

C. When to trigger database placement?

The database placement policy presented above includes checking if more fragment replicas are needed and if their locations are optimal. In the Grid environment, two parameters can trigger the database placement: (i) access frequency, and (ii) underlying environment. The placement policy then uses these parameters to schedule and execute the placement activities. A possible solution is to monitor periodically the database fragment access frequency and the Grid environment. The monitoring periodicity can be modified according to the change levels of the two parameters. If, for example, during the last placement activities there was no action needed, we can decrease the monitoring periodicity. On the other hand, if the last placement activities show that more replicas are needed or the replicas must be re-allocated, we can increase the monitoring periodicity.

In order to reduce the overhead generated by frequently using the monitoring mechanisms, we can monitor only fragments that their access frequencies change and we can monitor only sites that their resources change.

V. EXPERIMENTAL ANALYSIS

In this section, we evaluate the performance of our proposed database placement policy with simulation experiments using OptorSim simulator [10]. First, we describe the simulation framework, and then we give the experiment results.

A. Simulation framework

To evaluate the performance of our database placement policy, we have developed an extension of the OptorSim simulator [10]. OptorSim is a data grid simulator package written in Java used to allow experimentations of both file access optimisation and data replication strategies. We have implemented a round robin database placement algorithm and our database placement policy. We assumed a given number of sites, database fragments and join queries. Queries are submitted across Grid sites. Each join query requires two or many database fragments. The transfer of data from one site to another incurs a cost corresponding to the size of the data divided by the bandwidth of the network link. The simulation parameters are defined in table I.

TABLE I
SIMULATION PARAMETERS

Parameter	Explanation	Value
$/GS/$	Number of sites	10
$/CE/$	Number of computing elements / site	100 .. 500
$/SE/$	Number of storage elements / site	100 .. 500
$SR(SE_i)$	Storage capacity / storage element	0 ..50 GB
$STAB(SE_i)$	Stability / storage element	0 .. 1
NB	Network Bandwidth	100 MB .. 1GB/s
NS	Network Start-up delay	6 .. 10μs
$/F/$	Number of fragments	100 .. 1000
$Size(F_k)$	Size of fragment	2 .. 500 MB
ir_{Fk}	Initial number of F_k replicas	1
$/Q/$	Number of queries	10 .. 80
Nj	Number of join operations	1 .. 5

The objective of simulation experiments is to evaluate the quality of data distribution and the effect of our database placement policy on the query processing in Grid environment. For each experiment, we computed:

- The average query communication cost including the fragment transfer cost and the join result transfer cost (bandwidth consumed);
- Average query response time including data transfer time, compute time, I/O time, and queue time.

The query communication cost represented by the amount of data transferred is important from the perspective of overall resource utilization.

B. Experiment results

We have performed three set of experiments to evaluate the performance of our database placement policy. The first set of experimentations evaluates the effects of the database placement on the query communication cost. We have conducted a series of experiments with different number of join operations (ranging from 1 to 5). For each experiment we fix the number of the storage elements (400 elements), the number of fragments (600 fragments) and the number of queries (80 queries). Table II shows the communication cost, in terms of fragment transfer cost over the network and join result transfer cost, induced by an execution of 80 queries using two data placement alternatives (round robin and our placement policy).

TABLE II
COMMUNICATION COST FOR DIFFERENT NUMBER OF JOIN OPERATIONS

Number of join operations		1	2	3	4	5
Round robin placement C_1	Communication cost	101	204	367	626	871
	Placement transfer cost	0	0	0	0	0
	Placement search cost	0	0	0	0	0
	Total cost (sec)	101	204	367	626	871
Our placement policy C_2	Communication cost	27	38	67	119	174
	Placement transfer cost	22	59	88	145	178
	Placement search cost	9	9	10	11	12
	Total cost (sec)	58	106	165	275	364
C_1 - C_2		+43	+98	+202	+351	+507

When the round robin placement is used, the distribution of fragments across the Grid generates higher communication costs. When we introduce our database placement policy, query processing performs lower communication cost. Clearly, dynamic fragment replication and re-allocation decided by our database placement policy helps to reduce significatively the communication cost, but it causes additional costs (placement transfer and search costs). Note that the fragment replication performed will also reduce communication cost of future queries. We remark that our approach outperforms the round robin data placement. This is clearly pointed by table II. Since in spite of the two added costs, the total cost of our policy is lower than that of the round robin data placement. Moreover, we can still improve our total cost by decreasing the size of transferred fragments by using techniques of local reducing like semi-join operations [6].

Figure 2 shows the variation of the communication cost induced by the execution of 80 queries with different number of join operations. The results are the average over the series of experimentations performed for the two alternatives (round robin and our placement policy).

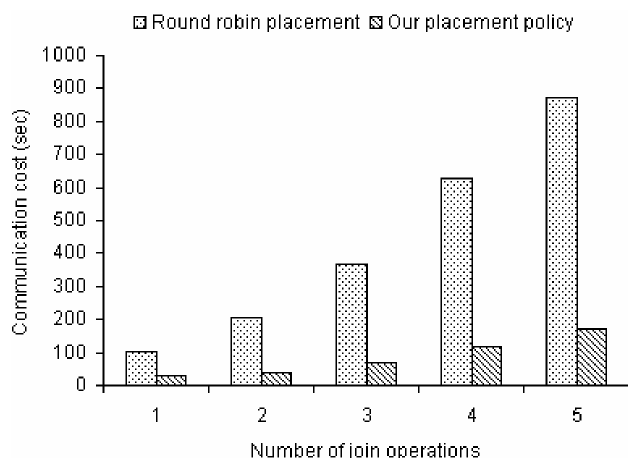

Fig. 2 Communication cost generated by the round robin and our placement policy

The second set of experiments studies the variation of the query communication cost for different number of join operations. Through these experimentations, we want to know how our database placement policy reacts when we execute queries containing a combination of join operations. We have conducted a series of experimentations with various number of join operations (ranging from 1 to 5). For each experiment, we compute the query communication cost induced by an execution of a number of queries (ranging from 10 to 80). To focus on network resource usage, Figure 3 shows the variation of query communication cost (total amount of data transferred) for different number of queries.

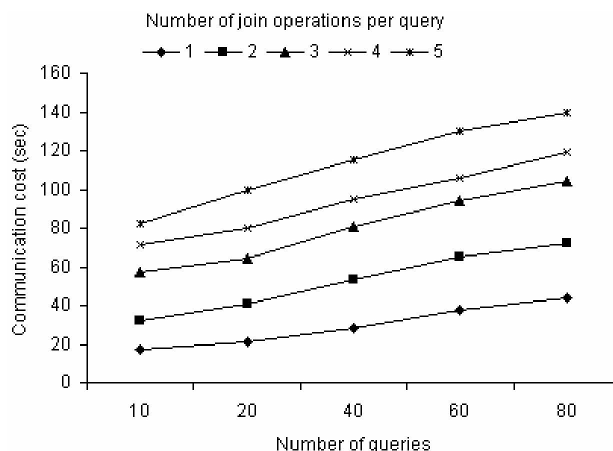

Fig. 3 Communication cost for different number of queries

We remark that when we increase the number of queries and the number of join operations, the difference in communication cost is very small. We justify this by the ability of our database placement policy to dynamically place fragments over the Grid. As seen in Figure 3, our database placement policy performs a good network resource utilization.

The third set of experimentations studies the variation of the query response time for different number of queries. Figure 4 shows the average response time for the system parameters of Table I. Clearly, dynamic database placement helps to enable load sharing and to reduce query response time. This figure shows that dynamic database placement represents a “best possible” choice of database placement.

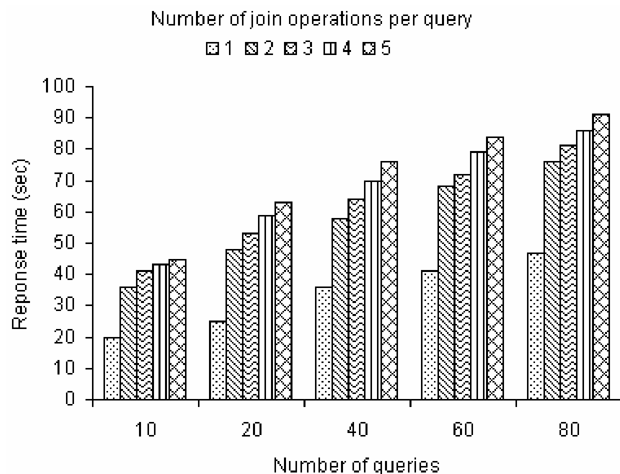

Fig. 4 Query response time for different number of queries

Hence, we can conclude that dynamic database placement integrated with a query optimizer leads to reduce the communication cost and the response time.

VI. Conclusion

We addressed the problem of query processing and database placement in a Grid environment. This paper showed how a query optimizer can be extended with a database placement policy so that it produces good query execution plan and implicitly makes long-term placement decisions. We have proposed a dynamic database placement policy which can be integrated into a query optimizer. We have used a Data Grid simulator, OptorSim, to evaluate the proposed database placement policy. Our experimental results show that dynamic database placement leads to accurate performances.

For future works, we plan to implement the proposed database placement policy on real Grid environment and to use some techniques, like parallel semi-join operations, to reduce the size of transferred fragments.

References

[1] M. T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, 2nd ed. Prentice-Hall, 1999.

[2] J. Smith, A. Gounaris, P. Watson, N. W. Paton, A. Fernandes and R. Sakellariou, “Distributed Query Processing on the Grid,” in *Proceedings of the Third Workshop on Grid Computing, GRID2002*, Baltimore, MA, 2002, pp. 380-387.

[3] S. Narayanan, U. Catalyurek, T. Kurc, X. Zhang and J. Saltz, “Applying Database Support for Large Scale Data Driven Science in Distributed Environments,” in *4th International Workshop on Grid Computing (Grid2003)*, Phoenix, Arizona, November 2003, pp. 141-149.

[4] A. Gounaris, N. W. Paton, R. Sakellariou and A. A. Fernandes, “Adaptive Query Processing and the Grid: Opportunities and Challenges,” in *DEXA Workshops*, Zaragoza, Spain, August-September 2004, pp. 506-510.

[5] H. Ye, B. Kerhervé and G. von Bochmann, “Revisiting Join Site Selection in Distributed Database Systems,” in *Euro-Par 2003, Parallel Processing, 9th International Euro-Par Conference*, ser. Lecture Notes in Computer Science, vol. 2790. Klagenfurt, Austria: Springer-Verlag, August 2003, pp. 342-347

[6] C.H. Lee and M.S. Chen, “Distributed Query Processing in the Internet: Exploring Relation Replication and Network Characteristics,” in *ICDCS '01: Proceedings of the 21st International Conference on Distributed Computing Systems*, Washington, DC, USA, 2001, pp. 439-446.

[7] H. Stockinger, O. F. Rana, R. Moore and A. Merzky, “Data Management for Grid Environments,” in *High-Performance Computing and Networking, 9th International Conference, HPCN Europe 2001*, ser. Lecture Notes in Computer Science, vol. 2110, Amsterdam, The Netherlands: Springer-Verlag, June 2001, pp. 151-160.

[8] T. Kosar and M. Livny, “Stork: Making Data Placement a First Class Citizen in the Grid,” in *Proceedings of 24th IEEE Int. Conference on Distributed Computing Systems, ICDCS 2004*, Tokyo, March 2004.

[9] F. Bonnassieux, R. Harakaly and P. Primet, “Automatic Services Discovery, Monitoring and Visualization of Grid Environments: The MapCenter Approach,” in *First European Across Grids Conference*, Santiago de Compostela, Spain, February 2003, pp. 222-229.

[10] D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, C. Nicholson, K. Stockinger, and F. Zini, “Optorsim: A simulation tool for scheduling and replica optimisation in data grids,” in *Proceedings of Computing in High Energy Physics, CHEP 2004*, Interlaken, Switzerland, 2004.