International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:1, 2008

A Graph-Based Approach for Placement of No-Replicated Databases in Grid

Cherif Haddad, and Faouzi Ben Charrada

***Abstract*—**On a such wide-area environment as a Grid, data placement is an important aspect of distributed database systems. In this paper, we address the problem of initial placement of database no-replicated fragments in Grid architecture. We propose a graph based approach that considers resource restrictions. The goal is to optimize the use of computing, storage and communication resources. The proposed approach is developed in two phases: in the first phase, we perform fragment grouping using knowledge about fragments dependency and, in the second phase, we determine an efficient placement of the fragment groups on the Grid. We also show, via experimental analysis that our approach gives solutions that are close to being optimal for different databases and Grid configurations.

***Keywords*—**Grid computing, Distributed systems, Data resources management, Database systems, Database placement.

I. Introduction

GRID architectures enable the use of computing, data and communication resources distributed in a wide-area environment [9]. One of the most challenges for the using of such distributed environment is the mapping of the data to storage space available on Grid [6, 7]. The ways in which data is distributed across sites in Grid architecture have a significant effect on the performance of a distributed database system. In fact, a good data placement can enhance efficient computation, reduce access time and minimize overall usage of resources [1, 4, 5]. Distributed Database Systems provide facilities to manage datasets in the context of Grid environment [8]. A number of Grid characteristics (heterogeneity, high scale and dynamicity) make Grid database placement difficult.

Generally, database placement is performed according to the access patterns. Initially, we perform a database no-replicated fragments placement without any information about access patterns. This database initial placement is important because it will reduce the number of fragment reallocations over the network when database fragments are queried. For the database initial placement, we don't have any knowledge on access patterns. The key for a good database initial placement is enhancing efficient computation, reducing access time and minimizing overall usage of resources.

Ch. Haddad is with the Department of Computer Science, Faculty of Sciences of Tunis, 1060 Tunis, Tunisia (corresponding author 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).

In this paper, we are interested in the problem of the initial placement of relational database no-replicated fragments in Grid architecture. It consists of determining where to place a given set of database fragments on a network of computing sites in order to optimize the use of computing, storage and communication resources. For this purpose we propose a graph-based approach to resolve the data initial placement of relational databases on a Grid. We assume a database schema, a set of information about fragments dependency, and information about Grid sites and networks. In this work, we compare our approach to the Round-Robin and the Hashing placement techniques [2].

The remainder of this paper is organized as follows. Section 2 outlines the parameters considered in our placement approach. Section 3 presents our proposed database initial placement approach. Section 4 describes some experiments conducted to evaluate our approach. Finally, section 5 concludes our paper.

II. Grid Environment

We model a Grid as a set of sites, each comprising a number of computing and storage elements. Each site can have a different number of computing elements, a different number of processors and hence different computing capabilities. We assume that all processors have more or less the same performance. Sites are connected to each other by WAN's limited bandwidth and computing elements within a site are joined together over a local area network.

The parameters considered in our initial database placement decision are the following:

1. *Site parameters*: Each Grid site is denoted by GS_a and for each site GS_a, $LANBW(GS_a)$ represents the local area network bandwidth of GS_a. $PROC(GS_a)$ represents the computing capability of the site GS_a measured by the number of processors;
2. *Storage element parameters*: Each storage element is denoted by SE_i and for each storage element SE_i, $SR(SE_i)$ represents the space reserved to store database fragments, $DISKBW(SE_i)$ represents the disk bandwidth of SE_i, $STAB(SE_i)$ represents the stability of SE_i which encompasses storage element failures, communication failures and the disconnection of the storage element from the grid. The storage element stability is expressed as the average probability of a storage element being up;

3. *Network parameters*: Sites are linked together through a communication network. In our work, the communication cost $CC(GS_a, GS_b)$ 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* no-replicated fragments $\{F_1, F_2, ..., F_m\}$. For each fragment F_j, $Size(F_j)$ represents the size of fragment F_j. We assume that fragmentation of database relations has been carried out before the placement phase. While fragmentation is an important issue, our main concern here is how the fragments should be placed around the Grid.

III. Proposed Approach

In this section, we present our proposed approach to place database no-replicated fragments across a set of sites. We first explain the graph-based approach and then we give our placement algorithms.

A. Graph-Based Placement Approach

Our approach proceeds in two phases: in the first phase, we perform fragment grouping using knowledge about fragments dependency and, in the second phase, we determine an efficient placement of the fragment group on the Grid. The grouped fragments are represented by a set of dependency graphs. Our approach ensures:

- Fragments that tend to be used together in answering queries should be placed together;
- By grouping fragments into groups represented by graphs before placement, we reduce the search space of placement problem.

In the following, we give some notions that are needed to define our placement approach. For the initial placement, we consider that we have knowledge on the fragment dependency. This knowledge can be produced at the database design phase [4]. The fragment dependency can be represented by a matrix $|F|\times|F|$ $[d_{i,j}]_{m\times m}$ constructed as follows. For $1 \le i \le m$, $1 \le j \le m$, $d_{i,j}=1$ if fragment F_i depends on F_j, otherwise $d_{i,j}=0$. The fragment dependency matrix is used to generate the fragment dependency graphs $G_k = (V,E,p)$, where V is the vertex set of G_k (fragments), E is the edge set (dependencies between fragments) and p assigns to each edge $e_{i,j} = (v_i, v_j)$, $i \neq j$, a value $p(e_{i,j}) = d_{i,j}$.

For each $v_i \in V$, the fragment dependency degree of v_i, e.g., $d(v_i)$, is equal to the total number of dependencies of both fragments v_i and v_j, where $(v_i, v_j) \in E$, $i \neq j$:

$$d(v_i) = \sum_{v_j \in V; i \neq j} p(v_i, v_j) \quad (1)$$

In our work, we suppose that the movable unit is not a fragment but a group of fragments. Using this assumption, we compute the total size of fragments represented by a dependency graph G_k. This metric is computed as follows:

$$Size(G_k) = \sum_{v_i \in V} Size(v_i) \quad (2)$$

B. Initial Placement Algorithms

Scalability is an important concern for our placement approach. To narrow the search field, our database placement is defined by two stages. The first one consists to choose candidate sites; this choice considers sites network bandwidths and sites computing capabilities. The second stage concerns the selection of storage elements inside chosen sites; this selection is made according to storage element's stability, disk bandwidth and space reserved to store database no-replicated fragments.

The main behavior of the initial placement algorithm can be resumed as follows. First, fragments are grouped according to their dependency as defined in section III.*A*. The set of grouped fragments is partitioned into a list of disjoint subsets $\langle G_1, \ldots, G_k \rangle$ in a decreasing order of $Size(G_k)$. Then, for each G_k we choose a site with highest network bandwidth and highest computing capability. The Grid initial placement algorithm is given by algorithm 1.

Algorithm 1 Grid Initial Placement

Require: *Grid*: Grid environment,
$\{G_k\}$: Set of grouped fragments
Ensure: Placement *P*
1. $P = \varnothing$
2. Let $G = \{G_k\}$ /* Queue of grouped fragments sorted in a decreasing order of $Size(G_k)$ */
3. **For all** G_k in G **do**
4. Generate a set $L_{Gk} = \{GS_a\}$ of sites sorted in a decreasing order of $\langle LANBW(GS_a), PROC(GS_a) \rangle$
5. **While** (There exist fragments in G_k to be placed) **do**
6. $GS = RemoveItem(L_{Gk})$
7. $P_{GkGS} = SiteInitialPlacement(G_k, GS)$
8. $P = P \cup P_{GkGS}$
9. **End While**
10. **End For**

Output: *P*

The time complexity of the Grid initial placement algorithm is $O(ks)$ where k is the number of fragment groups and s is the number of sites in the Grid.

The Grid initial placement just determines the site GS_{Gk} where the group G_k has to be placed. The decision where the fragments of G_k has to be placed is made by the site initial placement. The site initial placement algorithm is given by algorithm 2.

Due to the fact that stability of storage elements $STAB(SE_i)$ can vary dynamically, storage elements with a high stability are advantaged. For each G_k we try to place it on the first storage element of the list. If the selected storage element can't

provide enough storage to place fragments of G_k, we compact the group by excluding a fragment with minimum dependency degree. If there are many fragments we choose the fragment with minimum size.

Algorithm 2 Site Initial Placement

Require: GS_a: Site, $\{G_k\}$: Set of grouped fragments
Ensure: Placement P_{GkGS}

1. $P_{GkGS} = \varnothing$
2. Generate a set $SE_{GSa} = \{SE_i\}$ /* of candidate storage elements of site GS_a sorted in a decreasing order of $\langle STAB(SE_i), DISKBW(SE_i), SR(SE_i) \rangle$ */
3. **While** (There exist fragments in G_k to be placed) **do**
4. $SE_i = GetFirstItem(SE_{GSa})$
5. **If** $SR(SE_i) >= Size(G_k)$ **then**
6. $P_{GkGS} = P_{GkGS} \cup \{(G_k, SE_i)\}$
7. $SR(SE_i) = SR(SE_i) - Size(G_k)$
8. **Else**
9. $F = Compact(G_k)$ /* F is the fragment with minimum dependency degree in G_k */
10. **While** (F is not placed) **do**
11. $SE_i = GetNextItem(SE_{GSa})$
12. **If** $SR(SE_i) >= Size(F)$ **then**
13. $P_{GkGS} = P_{GkGS} \cup \{(F, SE_i)\}$
14. $SR(SE_i) = SR(SE_i) - Size(F)$
15. $Exit()$
16. **End If**
17. **End While**
18. **End If**
19. **End While**

Output: P_{GkGS}

The time complexity of the site initial placement algorithm is $O(n^2)$ where n is the average number of storage elements per site. Hence, the overall time complexity to obtain the final placement of fragments is $O(ksn^2)$, where k is the number of fragment groups and s is the number of sites in the Grid.

IV. Experimentations

To test the performance of our approach, we implemented a data placement prototype which gave us the opportunity to compare the results of our approach to those given by round robin and hashing placement techniques [2]. This prototype allows us to compute the variation of a placement algorithm running cost (search cost and data fragment transfer cost) according to the number of sites and the number of fragments. Also, it allows us to evaluate the quality of the data distribution and the performance gain when fragments are queried. To implement our prototype, we have developed an extension of OptorSim simulator. OptorSim is a Data Grid simulator package written in Java ™ [3]. Using this prototype, we have run our initial placement algorithm and other placement strategies on a wide range of different databases and Grid configurations.

Three sets of experiments are performed. The first investigates the relationship between algorithm running cost (search cost and transfer cost) and Grid configurations. The second one shows the impact of the database configuration on a given data placement method. The third evaluates the effect of the initial placement on the query communication cost.

Table I shows the running cost of the three placement approaches (the Round-Robin placement, the Hashing-Share placement technique [2] and our approach) for different Grid configurations (number of storage elements ranging from 100 to 500).

From this table, we remark that the running cost of the three approaches increases with the number of storage elements for a given database configuration. The search cost is influenced by the number of storage elements because more elements take more time to analyse. We remark that the running cost of our approach is always the less one.

Table II shows the variation of running cost for different database configurations (number of fragments ranging from 100 to 1000). We remark that our algorithm gives a lower running cost compared to the two other approaches. In the other hand, we notice that the running cost of our approach does not increase exponentially when the number of fragments increases linearly.

Since we want to evaluate the quality of the data distribution, we computed the communication cost when fragments are used in join queries. The communication cost consists of two components: the fragment transfer cost and the result transfer cost of the join operations. In order to simulate a real Grid network, we use different network bandwidths (ranging from 100 MB/s to 1 GB/s) and startup-delays (ranging from 6 to 10 μs) to compute the query communication cost.

Fig. 1 shows the variation of the query communication cost for different number of queries. We have run our initial placement algorithm, the Round-Robin and the Hashing-Share strategies on a wide range of different databases and Grid configurations. We note that the communication cost increases with the number of queries.

We remark that when the number of queries increases, the fragments become more and more dependent. We note that our approach preserves the dependency between fragments used together in join operations by placing depended fragments in the same site. As a result, the cost generated by the correlation constraint will be reduced and will prevent the query communication cost from increasing exponentially.

TABLE I
RUNNING COST FOR VARIOUS NUMBER OF STORAGE ELEMENTS

		Running cost (sec)				
Number of storage elements		100	200	300	400	500
Round-Robin	Search cost	10	13	15	17	18
	Transfer cost	2208	2424	2557	2602	2629
	Total cost	2218	2437	2572	2619	2647
Hashing-Share	Search cost	12	16	17	18	20
	Transfer cost	2196	2460	2564	2620	2674
	Total cost	2208	2476	2581	2638	2694
Our approach	Search cost	7	8	8	9	10
	Transfer cost	1607	1914	2048	2089	2113
	Total cost	1614	1922	2056	2098	2123

TABLE II
RUNNING COST FOR VARIOUS NUMBER OF FRAGMENTS

		Running cost (sec)					
Number of fragments		100	200	400	600	800	1000
Round-Robin	Search cost	4	7	13	17	22	28
	Transfer cost	438	836	1692	2602	3384	4246
	Total cost	442	842	1705	2619	3406	4274
Hashing-Share	Search cost	7	8	15	18	25	35
	Transfer cost	434	844	1740	2620	3442	4436
	Total cost	441	852	1755	2638	3467	4471
Our approach	Search cost	3	4	6	9	11	14
	Transfer cost	280	648	1221	2089	2601	3440
	Total cost	283	652	1227	2098	2712	3453

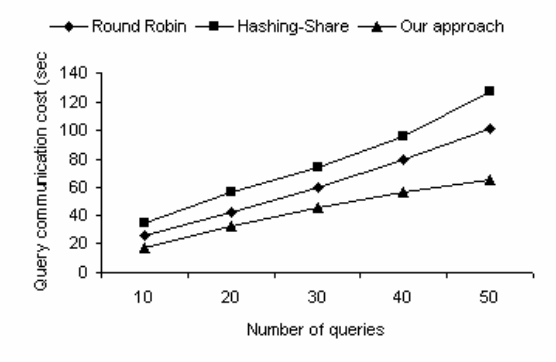

Fig. 1 Query communication cost for various number of queries

V. CONCLUSION

We have presented a graph-based approach to the initial placement of database no-replicated fragments in the context of Grid environment. Our approach is graph-based and uses the connectivity between fragments to place the maximum of connectivity on the same site. The main characteristic of our approach is to preserve the connectivity between fragments in order to reduce both execution time and communication cost between sites. It is clear that the objective of our approach is not to find the best initial placement of no-replicated fragments but to define an initial placement that can reduce the movement of fragments during their use through join queries.

As future works, we plan to study the effect of the replication on the query processing cost.

REFERENCES

[1] I. D. Baev and R. Rajaraman. Approximation algorithms for data placement in arbitrary networks. In *SODA'01: Proceedings of the twelfth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 661–670, Philadelphia, PA, USA, 2001.

[2] A. Brinkmann, K. Salzwedel, and C. Scheideler. Compact, adaptive placement schemes for non-uniform capacities. In *Proceedings of the 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 53–62, Winnipeg, Manitoba, Canada, August 2002.

[3] 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.

[4] Y. Huang and J. Chen. Fragment allocation in distributed database design. *Journal of Information Science and Engineering*, 17(3):491–506, 2001.

[5] Y. Huang and N. Venkatasubramanian. Data placement in intermittently available environments. In *High Performance Computing - HiPC 2002, 9th International Conference*, volume 2552 of *Lecture Notes in Computer Science*, pages 367–376, Bangalore, India, December 2002. Springer-Verlag.

[6] 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,(ICDCS2004)*, Tokyo, March 2004.

[7] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *International Symposium of High Performance Distributed Computing, HPDC-11*, Edinburgh, Scotland, July 2002.

[8] H. Stockinger. Distributed database management systems and the data grid. In 18th IEEE Symposium on Mass Storage Systems and 9th NASA Goddard Conference on Mass Storage Systems and Technologies, San Diego, CA, April 17-20 2001.

[9] H. Stockinger, Omer F. Rana, R. Moore, and A. Merzky. Data management for grid environments. In *High-Performance Computing and Networking, 9th International Conference, HPCN Europe 2001*, volume 2110 of *Lecture Notes in Computer Science*, pages 151–160, Amsterdam, The Netherlands, June 2001. Springer-Verlag.