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 use 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.

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 $G_S$, and for each site $G_S$, LANBW($G_S$) represents the local area network bandwidth of $G_S$. PROC($G_S$) represents the computing capability of the site $G_S$ measured by the number of processors;

2. Storage element parameters: Each storage element is denoted by $SE$, and for each storage element $SE$, SR($SE$) represents the space reserved to store database fragments, DISKBW($SE$) represents the disk bandwidth of $SE$, STAB($SE$) represents the stability of $SE$ 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;

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: echarrada@gnet.tn).
3. **Network parameters**: Sites are linked together through a communication network. In our work, the communication cost \( CC(GS_G, GS_S) \) between two sites \( GS_G \) and \( GS_S \) 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 \( F_i \), \( Size(F_i) \) represents the size of fragment \( F_i \). 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 \( 1 \leq i \leq m \), \( 1 \leq j \leq m \), \( d_{i,j}=1 \) 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 \( G_i = (V,E,p) \), where \( V \) is the vertex set of \( G_i \) (fragments), \( E \) is the edge set (dependencies between fragments) and \( p \) assigns to each edge \( e_{ij} \), \( i \neq j \), a value \( p(e_{ij}) = d_{i,j} \).

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 \( (v_i,v_j) \in E, i \neq j \):

\[
d(v_i) = \sum_{v_j \in V, i \neq j} p(v_i,v_j) \tag{1}
\]

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 \( G_i \). This metric is computed as follows:

\[
Size(G_i) = \sum_{v_i \in V} Size(v_i) \tag{2}
\]

#### 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 \( \{G_1, ..., G_k\} \) in a decreasing order of \( Size(G_i) \). 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. \( P = \emptyset \)
2. Let \( G = \{G_k\} \quad */ 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 \( LS = \{GS_G\} \) of sites sorted in a decreasing order of \( LANBW(GS_G), PROC(GS_G) \)
5. While ( There exist fragments in \( G_k \) to be placed ) do
6. \( GS = RemoveItem(LS) \)
7. \( P_{GGS} = SiteInitialPlacement(G_k, GS) \)
8. \( P = P \cup P_{GGS} \)
9. End While
10. End For

**Output**: \( P \)

The time complexity of the Grid initial placement algorithm is \( O(k \cdot s) \) 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 \( GS_G \) 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 \( STAB(SE) \) 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:** \( G_S \); Site, \( \{ G_k \} \); Set of grouped fragments

**Ensure:** Placement \( P_{GGS} \)

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

**Output:** \( P_{GGS} \)

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 \(^{3M} \) [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 \( \mu s \)) 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

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

TABLE II

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

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