Reducing Fragment Oscillation of Dynamic Fragment Allocation in Non-Replicated Distributed Database System

Tarun Gulyani and Pallath Paul Varghese
Pitney Bowes Software
India

Abstract—Production and consumption of data has seen an exponential increase in recent times thus necessitating a solution which addresses the issues related to physical storage, reliability and accessing speed limits of centralized system. Distributed Database System is the solution, which overcomes these limits [1]. Designing Distributed Database System has many issues, and one of them is fragment allocation/movement. Various fragment allocation algorithms already exist in Distributed Database System. There are strengths and shortcomings of these algorithms. This paper gives the brief overview of these algorithms and proposes one new algorithm (Reduced Fragment Oscillation Algorithm). RFO algorithm moves the fragment from source node to target node by considering both the frequency of fragment access by region as well as individual nodes. It increases the overall system performance. RFO algorithm also reduces the amount of topological data required in decision making.

1. Introduction

Distributed Database System is the Database System in which data is stored at multiple computers (node) either placed at the same physical location or spread over several geographical locations. These computers are connected to some network, via internet, intranet or extranet. Demands of Distributed databases are increasing rapidly than ever before, because it overcomes the limitations of centralized system [1]. Distributed database hides the distribution of logical and physical components of databases from users. The prime motivation in distributed database systems are to improve performance, to increase the availability, expandability and access facility of data [8]. In Distributed database the major challenge is how to distribute the data in the best manner. If the architecture of distributed database is not perfect, then its performance will be worse in comparison to centralized database architectures. Although distributed systems are highly desirable, the heterogeneity and lack of adherence to standards, makes it difficult to build a proper functioning system. Complexity in designing distributed database architecture is in maintaining multiple disparate systems instead of one big centralized system. Different design techniques are used to maintain the consistency in Distributed Database Architectures. Various design techniques used in Distributed Database System are replication, duplication, fragmentation, local autonomy, synchronous and asynchronous techniques etc. In case of replication technique any change in the database is replicated over all the databases stored over multiple nodes. This process is significantly time consuming and requires high network bandwidth. In duplication technique, a copy of the master database is regularly maintained in another database but it is a simple process compared to replication. In local autonomy technique, each node of distributed database can define their own policy. In asynchronous technique, a constraints of time, execution latencies and message latencies does not exist. On the other hand synchronous technique have both upper and lower bound for time constraint, execution latencies and message latencies. Depending upon the requirement for distributed databases, these technologies are used. Major concern in Distributed Database System is how to architect a good design for fragmentation of data and data allocation [2,9]. Data fragmentation as the word suggest is to distribute the data into fragments over multiple nodes either physically co-located or present at several geographical locations. Data fragmentation technique helps increase the efficiency of data access/query performance, as it enables a mechanism to store the fragment of data at a node where it is frequently used. It increases the parallelism, because large transaction can be divided into sub transaction and all sub transaction start processing concurrently. Data fragmentation can be horizontal fragmentation, vertical fragmentation or mixed fragmentation [7]. Horizontal fragmentation can be achieved using either by range or hash fragmentation [3]. These fragmentation techniques work on static environment, in which data access pattern and/or query pattern is static. For dynamic environment, where access pattern change dynamically, static allocation decreases performance. Data fragmentation algorithms used in dynamic environment include Optimal algorithm [6], Threshold algorithm [11], NNA (Near Neighborhood Allocation) algorithm [4], BGBR algorithm [5], FNA (Fuzzy Neighborhood Allocation) algorithm [10] etc. Main objective of these algorithm is to minimize the access cost as well as data transfer cost in executing the set of queries. A brief overview of existing algorithms is presented in the following sections and various notations/terms used in the entire paper are listed in TABLE 1.
2. Existing Algorithms

In Optimal algorithm [6], a static environment method is used to initially distribute the fragments to all nodes in the distributed network and an optimal algorithm is executed at each node. This algorithm depends highly on the frequency of access. The transfer of fragments from one node to the other is influenced by the change in frequency of access of fragments by each node. And if the frequency of access of a fragment changes frequently then there is an increase in the cost of transfer of fragment and network traffic. This would also result into significantly high oscillations of fragments between nodes. In this algorithm the access time of the fragment with highest frequency of access is significantly reduced but has a negative effect on the access time for the other nodes. The reason being the transfer of the fragment between nodes does not take the network topology into consideration.

Storage space requirement of an Optimal algorithm [6] is more as each fragment has to store the access counter value corresponding to each node. The heuristic threshold algorithm [11] addresses this drawback. In the threshold algorithm a counter value at each fragment is initialized to 0. The counter value is incremented, whenever remote access to the fragment occurs, and its value is reset to 0 when local access to the fragment occurs. Counter value is incremented only at remote access and not at local access. When counter value of fragment exceeds the predefined threshold value, then the ownership of the fragment is transferred to the remote node that recently accessed the fragment. If the threshold value is t, then this algorithm assures that the fragment remains at the new node for at least t+1 accesses [11]. Threshold value plays a major role in this algorithm. If threshold value more, then migration of fragments at the node is less. But if threshold value is less, then migration of fragments at the node is more. Approach of this algorithm is heuristic, as the ownership of the fragment is transferred to the node which has recently accessed the fragment which may or may not be node which has highest access to the fragment. The algorithm does not use the knowledge of topology so optimal node chosen for migration may not be the globally optimal node.

Near Neighborhood Allocation (NNA) [4] algorithm is a variation of Optimal Algorithm where finding the node (say target node) to which the fragment needs to be transferred from the original node (say source node) is same as optimal algorithm. However the fragment is transferred to the nearest neighbor of the source node which is in the path between the source node and target node. Routing algorithm is used to find the nearest neighbor of the source node. Here the knowledge of network topology is taken into consideration while selecting the node for transfer of fragment. NNA algorithm avoids fragment oscillation, which occur in optimal algorithm and also avoids multihop transfers. This also helps in reducing the access time from the target node. The NNA performs better than Optimal Algorithm when the size of fragment and size of network is very large. However in NNA algorithm, by moving fragment from source node to neighboring node, decreases the delay in access time for only those nodes which are in the path of between source and target nodes but for others node this may increase. If fragment access count value of the nodes, which are not in the path between target and source nodes is much more than nodes in the path then performance of system decreases. NNA algorithm avoids oscillation, so it reduces the consumption of network resources. For small fragment size, optimal algorithm is better than NNA algorithm, because cost of movement of small fragment is not more than access cost.

An improved version of NNA algorithm called FNA (Fuzzy Neighborhood Allocation) algorithm [10] is able to detect oscillation conditions and provides a solution to prevent redundant fragment migration. FNA uses a fuzzy inference engine to detect oscillations in fragment requests and ignore fragment migrations. BGBR Algorithm [5] is an improvement on NNA algorithm. It reduces the response time and fragment migration time from source node to target node compared to NNA algorithm. BGBR algorithm requires the knowledge of the complete network topology for migrating fragment from source node to target node. It avoids fragment oscillation. Although NNA algorithm reduces oscillation by knowledge of topology but it does not use the complete knowledge of topology. In NNA algorithm for transfer of fragment from source node to neighboring node, the shortest path does not give the

<table>
<thead>
<tr>
<th><strong>Table 1</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NOTATIONS</strong></td>
</tr>
<tr>
<td><strong>Node</strong></td>
</tr>
<tr>
<td><strong>Source Node</strong></td>
</tr>
<tr>
<td><strong>Target Node</strong></td>
</tr>
<tr>
<td><strong>Neighbor Node</strong></td>
</tr>
<tr>
<td><strong>Region</strong></td>
</tr>
<tr>
<td><strong>Ni</strong></td>
</tr>
<tr>
<td><strong>F</strong></td>
</tr>
<tr>
<td><strong>A</strong></td>
</tr>
<tr>
<td><strong>R</strong></td>
</tr>
<tr>
<td><strong>Threshold t</strong></td>
</tr>
</tbody>
</table>
assurance of best path according to global topology. BGBR performs better in both small as well as large fragment sizes however NNA performs better only when size of fragment is large. BGBR reduces fragment mobility as compared to Optimal, Threshold and NNA algorithm. BGBR algorithm shows better result, but it requires significant amount of calculations and also it requires capturing of significant amount of data about network topology.

Ulus et. al. [12] has proposed an algorithm based on Markov Chain Model where the each node autonomously decides whether to transfer the ownership of a fragment in DDS to another node or not and it depends on the past accesses of the fragment. Each fragment continuously migrates from the node where it is not accessed locally more than a certain number of past accesses, namely a threshold value.

A new dynamic fragment allocation algorithm in Non-Replicated allocation scenario proposed by Nilarun [8], incorporates access threshold time constraint of database access and most importantly the volume of data transmitted to dynamically reallocate fragments to sites at runtime in accordance with the changing access probability of nodes to fragments .

The next section discusses the proposed algorithm which addresses few drawbacks in the existing approaches of Dynamic Fragment Allocation in Non-Replicated Distributed Database System.

3. Reduced Fragment Oscillation (RFO) Algorithm

The proposed algorithm (RFO Algorithm) for dynamic allocation of data in non replicated distributed database system is designed to address the issues in the existing approaches where the movement of the fragment depends only on the node which has the highest frequency of access to a fragment. But in RFO algorithm consideration is also given to the highest fragment accessing region instead of considering the target there by reducing the access time from more than one node.

Steps of Reduced Fragment Oscillation (RFO) Algorithm are:

1. Group all the nodes into regions, with each region having approximately equal number of nodes subject to the constraint that the nodes are in near proximity to each other.

2. Decide the value of m, through which whole network will divide in regions.

3. Calculate the physical location of database in degree term by projection on map.

Figure 1 shows the layout of the regions and the access patterns of each node in the region. The steps of grouping the nodes into different regions is given below:

Step 1.a: Decide the value of m, through which whole network will divide in regions.

Step 1.b: Calculate the physical location of database in degree term by projection on map.

Figure 2 shows Node x contains fragments $F_1$ to $F_i$, Node y contains $F_1$ to $F_j$ and Node z contains $F_1$ to $F_k$. Number of fragments at each node can vary. Each fragment can be accessed by local node and/or remote nodes. For example fragment $F_1$ of Node x is accessed locally by Node x as well as remotely accessed by Node y and Node z. A fragment can be accessed directly or indirectly by more than
one hops. Here fragment $F_2$ of Node $y$ is accessed by Node $x$ through Node $z$.

Step2: For each locally stored fragment initialize the Node-Fragment Access Counter matrix $A$ as $a_{xy}=0$, where $x$ is index of all fragments and $y$ is index of all nodes.

Step3: For each locally stored fragment initialize the Region-Fragment Access Counter matrix $R$ as $r_{xz}=0$, where $x$ is index of all fragments and $z$ is index of all regions.

Step4: Check access request for each stored fragment from both local node and remote nodes.

Step5: Fragment Access Counter matrix $A$, increases the access counter value for access of fragments from a node. If node $y$ access the fragment $x$ then $a_{xy}=a_{xy}+1$.

Step6: Region-Fragment Access counter matrix $R$, increases the access counter value for accessing fragments from the regions. If region $z$ access the fragment $x$ then $r_{xz}=r_{xz}+1$.

Figure 3 shows Node-Fragment Access Counter matrix $A$ is the size of $k$ by $n$, where $k$ denote the number of fragments and $n$ denotes the number of nodes.

Figure 4 shows Region-Fragment Access counter matrix $R$ is the size of $k$ by $m$, where $k$ denotes the number of fragments and $m$ denotes the number of regions.

Figure 5 shows Node contains local database $F$, which consist of more than one fragment. It contains Node-Fragments Access counter matrix $A$ and Region-Fragment Access counter matrix $R$.

Step7: Migrate the fragment from source to the target in the region which has max count in $R$ and then in that region move the fragment to the node in that region which has max count in $A$.

The algorithm only considers fragments at each node which has Region-Fragment Access counter value greater than a predefined threshold $t$.

Figure 6, 7 and 8 explains the process of identifying the target node for migrating the fragment.

Figure 6 shows $F_k$ is one of the fragment of Node N13 in Region 3, which is accessed by Nodes (N1,N2,N3,N4 and N5) from Region 1, (N6,N7,N8) from Region 2, (N9,N10) from Region M-1 and (N11,N12) from Region M.

Figure 7 shows only rows of Node-Fragment and Region-Fragment Access Counter matrix of Fragment $F_k$ according to access in Figure 6.
The Region-Fragment access count for each region is calculated. The region which has the maximum count is chosen as the region where the fragment would be moved. Region 1 has the highest access count to Fragment $F_k$ in comparison to other regions. Therefore Fragment $F_k$ moved to Region 1. In Region 1 the Node-fragment access count for all the nodes is computed for the Fragment $F_k$. The node which has the maximum count is chosen as the target node. Therefore (in the current example Figure 8) within Region 1, Fragment $F_k$ moved to Node N2.

If Optimal/Threshold algorithm is used instead of RFO algorithm then fragment $F_k$ will be moved to Node N11 in Region M. Because it has the maximum Node-Fragment access count of 150, which is greater than that of all other nodes.

Transferring the Fragment $F_k$ to Node N11 in Region M decreases the access time of node N11 for the fragment $F_k$, but increase the access time to all the nodes(N1,N2,N3,N4,N5) of Region 1. Resulting in decrease in total access time for the fragment $F_k$ from Region M, which had Region-Fragment access count of 155, but increasing the access time of the fragment $F_k$ from Region 1, which had Region-Fragment access count of 270.

RFO algorithm addresses this issue and therefore it reduces the global access time of fragment $F_k$ by transferring the fragment $F_k$ to node N2 instead of node N11.

## 4. Conclusion

Proposed algorithm decreases the migration/oscillations of fragments in comparison to optimal, and threshold algorithm using the knowledge of network topology which is not the case with both Optimal and Threshold. In comparison to NNA algorithm, RFO algorithm moves the fragment from source node to destination target node by considering both the frequency of fragment access by region as well as individual nodes, there by increasing the overall system performance. RFO algorithm also reduces the amount of topological data required in decision making in comparison to BGBR algorithm.

RFO algorithm depends on two parameters. One is the choice of threshold value which also directly impact fragment oscillation. Another parameter is division of whole network into m regions, where each region contains approximately equal nodes.

RFO Algorithm is suitable for Distributed database architectures where the access pattern to a fragment changes dynamically and distributed database is not replicated. This algorithm significantly minimizes the data transfer and also decreases the amount of complexity required in identifying a suitable target node.
References


