Rule Based Fragment Allocation in Distributed Database Systems

Mohammad Reza Abbasifard

MODB Lab., School of Computer Engineering,
Iran University of Science and Technology,
Tehran 1684613114,Iran

Abstract. Allocating data fragments in distributed database systems is an important issue in distributed database (DDB) systems. In this research work, we will show how rule based policy languages can be used to represent different data fragment allocation techniques. Results indicate that, using rule based languages like prolog can significantly simplify the representation of such algorithms for any further developments and optimizations. Our examples show that our approach can be extended to be used in different areas of distributed systems.

Keywords: Fragment Allocation, Rule Based Policies

1 Introduction

Developments in distributed algorithms, network technologies, and database theory in the past few decades led to advances in distributed database systems (DDS). A DDS is a collection of database nodes connected by a communication network, in which each node is a database system in its own right, but the nodes have agreed to work together, so that a user at any node can access data anywhere in the network exactly as if the data were all stored at the userś own node.

The primary concern of fragmentation in a DDS is to show how data should be divided and distributed among nodes in the underlying database. Fragmentation problem in a DDS is how to divide the data while allocation issue means how those fragments should be distributed over different DDS nodes. The data allocation problem, is NP-complete, and thus requires fast heuristics to generate efficient solutions [1]. Furthermore, the optimal allocation of database objects highly depends on the query execution strategy employed by a distributed database system, and the given query execution strategy usually assumes an allocation of the fragments.

A major cost in executing queries in a distributed database system is the data transfer cost incurred in transferring relations (fragments) accessed by a query from different nodes to the node where the query is initiated. The objective of a data allocation algorithm is to determine an assignment of fragments at different nodes so as to minimize the total data transfer cost incurred in executing a set

of queries. This is equivalent to minimizing the average query execution time, which is of primary importance in a wide class of distributed conventional as well as multimedia database systems.

An optimal, but not practical, solution for fragment allocation in DDS has been appeared in [2]. There are also a few fragment allocation algorithms [3–8] that are proven to be practical and show a reasonable performance. Several surveys of those algorithms are provided by [9–12]. Since all of these fragment allocation algorithms are expressed and implemented by imperative programming languages, they are usually difficult to understand and configured.

In this paper, using declarative rule based languages, we propose a novel technique that can be used to represent fragment allocation algorithms. In our technique, we consider fragment allocation strategy as a rule-based policy, implemented in a logic programming framework. The declarative representation of fragment allocation algorithms results in two major benefits: (1) since declarative representation of algorithms are much simpler than imperative ones, these algorithms can be changed and improved simpler when they are represented by rule-based languages; (2) the reasoning components of these algorithms can be relied on logic programming frameworks, and thus we will have simpler implementation of fragment allocation components in DDS. This technique also can be used to improve existing DDS fragment allocation simulators [13].

The rest of this paper is structured as follows: in section 2 we will briefly review some of major parameters of fragment allocation problem, section 3 is about our representation technique and section 4 briefly explains the implementation of our prototype model. Finally section 6 is our conclusion.

2 Fragment Allocation Problem

Fragment and data allocation algorithms are categorized into two major groups: static and dynamic. In static fragment allocation algorithms, data allocation has been completed prior to the design of a database depending on some static data access patterns and/or static query patterns. However, dynamic fragment allocation algorithms can change the data fragment allocation automatically during the deployment of the database. In a dynamic environment where these probabilities change over time, the static allocation solution would degrade the database performance.

Depending on the complexity of a data allocation algorithm, it may take the following parameters as inputs:

1. The fragment dependency graphs.
2. Unit data transfer costs between nodes.
3. The allocation limit on the number of fragments that can be allocated at a node.
4. The query execution frequencies from the nodes.

The fragment dependency graph models the dependencies between the fragments and the amount of data transfer incurred to execute a query. A fragment

dependency graph (as shown in figure 1) is a rooted directed acyclic graph with the root as the query execution site (Node Q in Figure 1) and all other nodes as fragment nodes (Node G, etc., in Figure 1) at potential nodes accessed by a query.

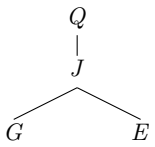

Fig. 1. A sample fragment allocation graph.

Assume that r_{ij} indicates the frequency of requirements by node i for fragment j, each fragment i is characterized by its size, n_i and t_{ij} indicates the cost for node i to access a fragment located on node j. Clearly, t_{ij} is a function of the following parameters:

- The average size of data fragments: s_j.
- The bandwidth of network link between i and j: w_{ij}.
- The delay of network link between i and j: d_{ij}.
- Other types of costs on network link between i and j, e.g. communication expenses: o_{ij}.

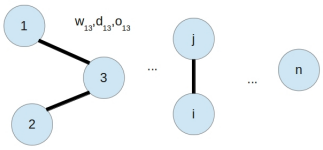

Fig. 2. A sample network parameters.

Therefore, users of the distributed database systems must be able to define t_{ij} for a fragment allocation algorithm based on the above mention parameters. Moreover, the frequency of the execution of each type k of the queries executed by node i on data item j, f_{ijk}, is another important factor for the fragment allocation algorithm. Note that, different types of database queries have different transfer costs. For instance, *select* (*se*) queries (specially those require joins on tables) may require large data transfers while *update* (*up*) and *delete* (*de*) queries do not require large data transfers. In fact, an efficient fragment allocation algorithm results in minimization of execution cost, which is shown in (1).

$$\sum_{k\in\{se,up,de\}} \sum_{i=1}^{m}\sum_{j=1}^{n} f_{ijk} \tag{1}$$

The distributed database allocation problem is to find the optimal placement of the fragments at the nodes. That is, we wish to find the placement, $P = \{p_1, p_2, p_3, \ldots, p_j, \ldots, p_n\}$ (where $p_j = i$ indicates fragment j is located at node i) for the n fragments so that the capacity of any node is not exceeded, that is shown in (2).

$$\sum_{i=1}^{m} r_{ij} n_j \leq c_{ij} \tag{2}$$

Moreover, the total transmission cost, shown in (3), should be minimized [8].

$$\sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij} t_{ij} \tag{3}$$

By restricting the use of the requirements matrix and having zero transmission cost, the distributed database allocation problem can be transformed to the bin packing problem, which is known to be NP-complete.

3 Methodology

In this paper, our goal is to develop a flexible and dynamic fragment allocation algorithm. Clearly, such algorithm must be considered as a distributed algorithms. Otherwise, adding a coordinator node can drastically decrease the flexibility of such algorithm. At the first glance, developing such distributed algorithm may look difficult as distributed logic programming and rule based frameworks are required for such algorithm. But, fortunately, this problem is not as difficult as what it looks. Because synchronizing the fragment allocation and its parameters, each node can act independently while we make sure the result of our executions for different nodes are same. Then, we just need to represent our fragment allocation algorithm using a rule based language and make sure the rules of each node and facts are properly synchronized.

```
delay(1,3,5).
...
reverse_bandwidth(1,3,0.5).
...
other(1,3,5).
```

Fig. 3. Representation of network as a set of facts.

In order to develop a fragment allocation algorithm in a rule-based language, first we need to represent above mentioned parameters as sets of facts. Then,

we need to develop our algorithm in terms of rules—similar to representation of policies using rule based languages. Obviously, the set of rules defining the fragmentation algorithm should be synchronized in each node as well.

The over all representation of network parameters in a rule based language is simple and natural. We can use simple sets of facts to represent s_j, w_{ij}, d_{ij}, and o_{ij}. For instance, Figure 3 shows that the delay between node 1 and 3 is 5 milliseconds, the reverse of the bandwidth is 05 1/mega-bytes, and the cost of communication for each mega-byte is 5 dollars. Then, t_{ij} can be computed as shown by (4), where γ_{ij} represents the user defined factors. This computation will be translated to a rule in our algorithm. Figure 4 shows a sample translation of such computation.

$$t_{ij} = \gamma_{ij} \times s_j \times w_{ij} \times d_{ij} \times o_{ij} \tag{4}$$

```
transfer_cost(I,J,T) :- user_defined_parameter(I,J,U),
                        size(J,S),
                        reverse_bandwidth(I,J,W),
                        delay(I,J,D),
                        other(I,J,O),
                        T is U*S*W*D*O.
```

Fig. 4. Representation of the computation of t_{ij} in our algorithm.

Similarly, the execution statistics, f_{ijk} can also be generated as a set of fact by the execution engine of DDS. The pre-defined parameter to show the execution cost of query type k on node i for the fragment j, e_{ijk}, is also defined as a fact by users. Therefore, for the simplest fragment allocation policy, where fragments are moved if the execution cost is larger than fragment relocation cost. In such algorithm, the trigger for moving the data item j from i_1 to i_2, $move_{i_1 i_2 j}$, can be computed through the following rule:

$$move_{i_1 i_2 j} \longleftarrow \sum_{k \in \{se,up,de\}} f_{i_1 j k} \leq r_{i_1 j} t_{i_1 j} \ \wedge \ \sum_{k \in \{se,up,de\}} f_{i_2 j k} > r_{i_2 j} t_{i_2 j} \tag{5}$$

Accordingly, this trigger runs two major events: physically moving the data item j from i_1 to i_2 and updating fragment allocation information in all of the nodes. Using rules of type (4) and (5), the inference engine needs to respond to the query (6), where X, Y, and Z are variables bound by inference engine. The result of such query will be used to activate triggers.

$$? - \ move_{X,Y,Z}. \tag{6}$$

Simply, one can use prolog *assert* and *retract* instructions in synchronization unit to update fragment allocation information. Based on this executions, the main procedure of fragment allocation component can be developed as shown in Figure 5.

```
1: function FRAGMENT_ALLOCATION
2:     while true do
3:         Run synchronization unit
4:         Update execution statistics
5:         if Any facts updated then
6:             Re-run the inference engine and query the move_{X,Y,Z} triggers.
7:             if There exists any trigger whose source is me then
8:                 Run the fragment transfer unit
9:             end if
10:        else
11:            Wait for synchronization period
12:        end if
13:    end while
14: end function
```

Fig. 5. The main procedure in fragment allocation component.

As mentioned before, rule based representation of fragment allocation algorithm makes those algorithms simple and easy to understand. For instance, let $a_{i_1 i_2}$ be a fact representing that there is a direct link between i_1 and i_2. Therefore, NNA [4] fragment allocation algorithm can be simply represented as

$$move_{i_1 i_2 j} \longleftarrow \sum_{k \in \{se,up,de\}} f_{i_1 j k} \leq r_{i_1 j} t_{i_1 j} \ \wedge \sum_{k \in \{se,up,de\}} f_{i_2 j k} > r_{i_2 j} t_{i_2 j} \ \wedge a_{i_1 i_2} \tag{7}$$

Similarly, FNA [5][3] and BGBR [6] parameters can be imported to our algorithms. Complicated reasoning for FNA also needs supporting Fuzzy logic resolutions and libraries by resolution frameworks.

4 Implementation

As mentioned in the previous section, in our approach, each node is considered as an independent system, synchronized with other nodes on fragment allocation mechanisms. Figure 6 shows the design of a node in our DDS. We are still working on the implementation of this project. The inference engine in our system will

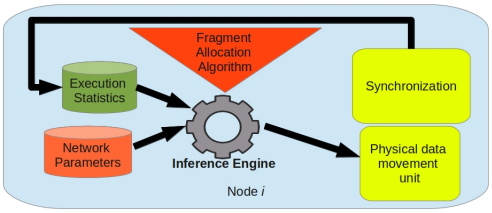

Fig. 6. Design of a single node in a DDS.

be XSB Prolog [14]. The implementation will be evaluated using the parameters introduced in [3, 4].

Synchronization is one of the most important components of our system. Synchronization is repeated in a period of time. The frequency of synchronization also depends on the speed of the execution of fragment allocation algorithm by inference engine. Apparently, each node must wait until receive the synchronization information from the rest of the nodes before each execution of the fragment allocation algorithm.

5 Conclusion

In this paper, we discussed a novel method for representing fragment allocation algorithms in a rule based system. Our results show that such representation makes a fragment allocation algorithm. The simplicity of the resulted algorithm can help one to extend existing algorithms and improve their performances. Moreover, the simplicity of the resulted algorithms eases configuring fragment allocation component in DDS.

We are planning to investigate using defeasible reasoning and argumentation theory [15][16] to extend our developments. Another promising direction for this research is to investigate other rule based system, e.g. Answer Set Programming [17][18] , and possibly get more speedups.

References

1. Meghini, C., Thanos, C.: The complexity of operations on a fragmented relation. ACM Trans. Database Syst. **16**(1) (March 1991) 56–87
2. Morgan, H.L., Levin, K.D.: Optimal program and data locations in computer networks. Commun. ACM **20**(5) (May 1977) 315–322
3. Basseda, R., Rahgozar, M., Lucas, C.: Fuzzy Neighborhood Allocation (FNA): A Fuzzy Approach to Improve Near Neighborhood Allocation in DDB. In: Advances in Computer Science and Engineering: 13th International CSI Computer Conference, CSICC 2008 Kish Island, Iran, March 9-11, 2008 Revised Selected Papers. Springer Berlin Heidelberg, Berlin, Heidelberg (2009) 834–837
4. Basseda, R., Tasharofi, S., Rahgozar, M.: Near neighborhood allocation (nna): A novel dynamic data allocation algorithm in ddb. In: 11th International Computer Society of Iran Computer Conference (CSICC2006)

5. Basseda, R., Rahgozar, M.: A novel fuzzy approach to improve near neighborhood allocation algorithm in ddb. In: 2009 IEEE/ACS International Conference on Computer Systems and Applications. (May 2009) 571–578
6. Bayati, A., Ghodsnia, P., Rahgozar, M., Basseda, R.: A novel way of determining the optimal location of a fragment in a ddbs: Bgbr. In: Systems and Networks Communications, 2006. ICSNC '06. International Conference on. (Oct 2006) 64–64
7. Ahmad, I., Karlapalem, K., Kwok, Y.K., So, S.K.: Evolutionary algorithms for allocating data in distributed database systems. Distributed and Parallel Databases **11**(1) (2002) 5–32
8. Corcoran, A.L., Hale, J.: A genetic algorithm for fragment allocation in a distributed database system. In: Proceedings of the 1994 ACM Symposium on Applied Computing. SAC '94, New York, NY, USA, ACM (1994) 247–250
9. Navathe, S., Ceri, S., Wiederhold, G., Dou, J.: Vertical partitioning algorithms for database design. ACM Trans. Database Syst. **9**(4) (December 1984) 680–710
10. Apers, P.M.G.: Data allocation in distributed database systems. ACM Trans. Database Syst. **13**(3) (September 1988) 263–304
11. Brunstrom, A., Leutenegger, S.T., Simha, R.: Experimental evaluation of dynamic data allocation strategies in a distributed database with changing workloads. In: Proceedings of the Fourth International Conference on Information and Knowledge Management. CIKM '95, New York, NY, USA, ACM (1995) 395–402
12. Basseda, R., Tasharofi, S.: Data allocation in distributed database systems. Technical report, University of Tehran: Technical Report No. DBRG. RB-ST (2005)
13. Basseda, R., Tasharofi, S.: Design and implementation of an environment for simulation and evaluation of data allocation models in distributed database systems. Technical report, University of Tehran: Technical Report No. DBRG. RB-ST (2005)
14. Swift, T., Warren, D.S.: Xsb: Extending the power of prolog using tabling. (2011)
15. Wan, H., Grosof, B.N., Kifer, M., Fodor, P., Liang, S.: Logic programming with defaults and argumentation theories. In Hill, P.M., Warren, D.S., eds.: Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings. Volume 5649 of Lecture Notes in Computer Science., Springer (2009) 432–448
16. Basseda, R., Gao, T., Kifer, M., Greenspan, S., Chell, C.: Representing flexible role-based access control policies using objects and defeasible reasoning. In Bassiliades, N., Gottlob, G., Sadri, F., Paschke, A., Roman, D., eds.: Rule Technologies: Foundations, Tools, and Applications - 9th International Symposium, RuleML 2015, Berlin, Germany, August 2-5, 2015, Proceedings. Volume 9202 of Lecture Notes in Computer Science., Springer (2015) 376–387
17. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R.A., Bowen, K.A., eds.: Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, August 15-19, 1988 (2 Volumes), MIT Press (1988) 1070–1080
18. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.T.: Potassco: The potsdam answer set solving collection. AI Commun. **24**(2) (2011) 107–124