Access Patterns Optimization in Distributed Databases Using Data Reallocation

Adrian Sergiu Darabant(✉), Leon Tambulea, and Viorica Varga

Department of Mathematics and Computer Science, Babes-Bolyai University, Cluj-Napoca, Romania
dadi@cs.ubbcluj.ro

Abstract. Large distributed databases are split into fragments stored on far distant nodes that communicate through a communication network. Query execution requires data transfers between the processing sites of the system. In this paper we propose a solution for minimizing raw data transfers by re-arranging and replicating existing data within the constraints of the original database architecture. The proposed method gathers incremental knowledge about data access patterns and database statistics to solve the following problem: online re-allocation of the fragments in order to constantly optimize the query response time. We model our solution as a transport network and show in the final section the experimental numerical results we obtain by comparing the improvements obtained between various database configurations, before and after optimization.

1 Introduction

Let us consider a distributed database with a set of fragments/shards F stored on a set of sites S in a communication network. A set Q of *applications/queries* is executed against the database. We start from the assumptions that in order to minimize the query execution time, the data transferred needs to be minimized. Thus the data allocation to the sites of the system needs to be implemented such that data transfer during query execution is minimized. This paper assumes that the fragments have been already determined and (eventually) allocated, and focuses on the problem of allocating (re-allocating) them in order to minimize the total cost of data transmission. In practice, the optimal initial allocation of fragments is not possible without apriori knowledge of the applications running on the database. Our solution relaxes this requirement by allowing an initial allocation unaware of the applications querying data. It then optimizes the allocation by observing, at the database level, the access patterns incurred by the queries and performing redundant re-allocation.

2 Related Work

Many aspects of the data allocation problem have been studied in the literature. Reid and Orlowska [6] studied the communication cost minimization problem

© Springer International Publishing AG 2017

DOI: 10.1007/978-3-319-64468-4_14

while replica allocation modeled as an integer linear programming with minimizing the execution cost has been approached in [1,3,6].

Menon in [7] considered non-redundant allocation. This paper focuses on redundant allocation. Wiese in [13] use clustering and clustered attributes. Huang and Chen in [5] propose a simple and comprehensive model that reflects transaction behavior in distributed databases.

The fragment allocation problem is NP-complete [9]. A genetic algorithm is proposed in [12], a genetic search-based clustering in [2] and an evolutionary approach in [4].

A reinforcement learning solution for allocating replicated fragments is presented in [8].

3 Query Evaluation and Data Transfer

Let $A(q)$ be the query evaluation tree for query q. We add a root node to this tree and we obtain a sub-tree rooted in the new node that corresponds to the entire query q (the new root represents the overall q query). Leaf nodes in $A(q)$ correspond to fragments, while internal nodes represent relational operators (unary or binary). When evaluating an operator op from q we get a transfer cost for data from the nodes where each operand of op is evaluated/stored to the node where op is evaluated.

In the next paragraphs we will use the following notations: $F = \{f_i | i = \overline{1,n}\}$ - fragment set of the database, $df_i = dim(f_i)$ - size of fragment $f_i, i = 1, n$, $S = \{s_i | i = \overline{1,m}\}$ - the sites of the system where fragments of F are stored, $S(f)$ - sites of the system where a fragment $f, f \in F$ is stored, $F(s)$ - fragments stored on site $s \in S$.

Starting from a predefined (current) state of the database (fragments, sites, current fragment allocation), in [10] we attach two values to each node of the query tree $A(q)$: d and c as follows: d - the size of data associated to the site (fragment size if the current node is a leaf node, or an estimation of the relational operator result size for internal nodes), and the costs vector $c = (c_1, \dots, c_m)$ (m is the number of sites) of evaluating the query on all sites. For leaf nodes this equates to the size of the fragment or zero. For an internal node corresponding to an operator op, c_i is the minimal cost of the required data transfers when the operator op is evaluated on site s_i. See our previous work [10].

The following paragraphs describe the computation method for the vector c in the case of the two possible cases: an unary and a binary operator.

Let op be an unary/binary operator and its current operand(s) $A(B)$ with its associated values: d_A and $c_A = [c_1^A, \dots, c_m^A]$. The incurred data transfer in the evaluation of op on site s_i depending on the location of operand(s) is given by the bellow expression (binop=1 for binary operators and 0 otherwise):

$$\begin{aligned} c_i = \min\{c_1^A + d_A, \dots, c_{i-1}^A + d_A, c_i^A, c_{i+1}^A + d_A, \dots, c_m^A + d_A\} \\ +binop \times \min\{c_1^B + d_B, \dots, c_{i-1}^B + d_B, c_i^B, c_{i+1}^B + d_B, \dots, c_m^B + d_B\} \end{aligned} \quad (1)$$

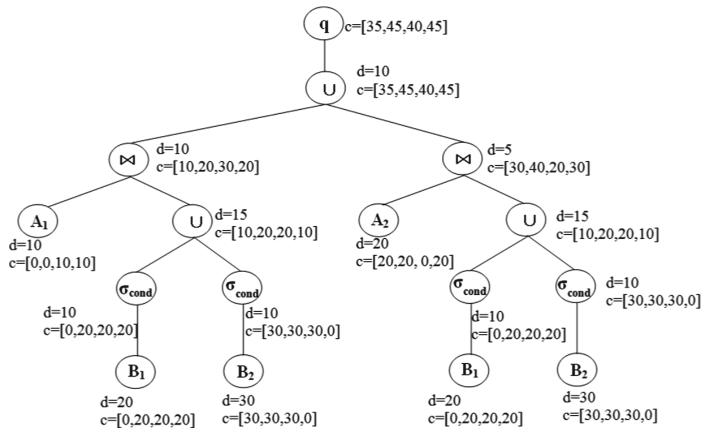

Fig. 1. The evaluation tree and the values associated to an example query.

We show in Fig. 1 the $A(q)$ tree built for some real values of the fragments size and results of the relational operators.

Using (1), we can compute the values for the vector c associated to query q - that is the root of the $A(q)$ tree in Fig. 1. Query q is executed on a specific site of the system. The vector c that labels the root of the $A(q)$ tree provides the minimal cost of the data transfers during the execution of query q. In the following we will analyze the required data transfer for the two possible cases: when f is a sub-tree of q, or f is used in a combination of unary operators. These cases are depicted in Figs. 2 and 3.

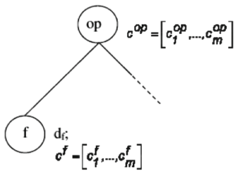
op $c^{op} = [c_1^{op}, ..., c_m^{op}]$
f d_f; $c^f = [c_1^f, ..., c_m^f]$

Fig. 2. Fragment used by a binary operator - one operand is always a leaf node (fragment)

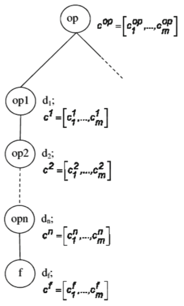
op $c^{op} = [c_1^{op}, ..., c_m^{op}]$
op1 d_1; $c^1 = [c_1^1, ..., c_m^1]$
op2 d_2; $c^2 = [c_1^2, ..., c_m^2]$
opn d_n; $c^n = [c_1^n, ..., c_m^n]$
f d_f; $c^f = [c_1^f, ..., c_m^f]$

Fig. 3. Fragment used by a sequence of unary operators

The two cases described above are valid for a sub-tree of the binary operator *op*. The same applies for the second sub-tree, corresponding to the second operand. If the fragment used by the second sub-tree is also stored on site s_i, then c_i^{op} will be null. If all fragments used by a query q are stored on the site s_i where the query is evaluated, then $c_i^{op} = 0$ for all operator nodes from $A(q)$.

Using the above analysis we can infer that by storing the fragment f on a site where the query accessing f is executed (let this site be s), we can reduce the data transfer cost by an amount r - equal to the size of fragment f or with the size of the result of the last unary operator applied to f, but before a binary operator applied to f - i.e. the first unary operator applied to f appearing strictly before a binary operator on f, if such exists. If fragment f appears multiple times in the evaluation tree $A(q)$ (as for example is the case for B_1 in Fig. 1), then the data transfer is reduced on all accesses to fragment f.

Given a distributed database and a time interval, we denote by Q the set of observed queries that are executed against the database and their access patterns in the given time period. Information about the access patterns is stored by the database's statistical module in views and can be retrieved for analysis. We should note that apriori knowledge about the database queries is not needed (as for the case of fragmentation). Instead we retrieve statistical observed information about operators, their evaluation and operator to query membership relations from the database statistics.

In order to speed up the evaluation of a query $q \in Q$ on a site $s \in S$, we can infer a set of *replication hints* for the fragments accessed by q, denoted as a triple (f, s, c) and signifying that by storing fragment f on site s we can reduce the data transfer cost by c. Since query q can be observed running a number $r, r \geq 1$, of times on site s, then by using the replicas according to the *replication hints* the data transfer costs are reduced by an amount of $r * c$. Considering the *replication hints* (f, s) proposed by all queries $q \in Q$ and the amount of reduction in data transfer cost for the resulting fragment storage policy we obtain a set of *replication hints* for Q denoted as:

$$P = \{(f_i, s_j, c_{ij}) | f_i \in F, s_j \in S, (f_i, s_j) \neq (f_k, s_l), \forall i \neq k \, and \, j \neq l\} \qquad (2)$$

In the following we assume that the database dictionary after an observed running interval contains information about: fragments, fragment allocation, queries and fragments accessed by a query. Suppose that this information is made available throughout computed views like an usual database would.

4 Induced Fragment Replication

Let $s_i \in S$ be a site containing some fragments of the database. Let ds_i be the available memory space on site s_i. We can only store new fragments within the limits of the available memory space. In the trivial case, if the available memory is infinite, the solution to the replication problems is total replication where the data transfer cost is null for any query. When the available memory space is limited our proposed replication model needs to find the optimal set of replicas within the memory space constraint such that data transfer cost is minimal for the overall set of queries in Q. v If the size of a fragment $f_i \in F$ is df_i and the available memory/storage space on site $s_j \in S$ is ds_j, then we propose a solution modeled as a transport network compatible flow problem.

A *replication hint* (f_i, s_j, c_{ij}) as mentioned in (2) has two possible implementation choices: to be retained/applied or dismissed. We introduce a new variable $r_{ij}, r_{ij} \in \{0, 1\}$ that denotes the above possibilities.

4.1 Network Flow Solution

When modeling the induced replication as a network flow problem we have two possible options: a global variant for the whole set S of sites, or individually for each site $s \in S$. We will describe the solution model for the former variant.

Given all the above described elements we propose a transport network denoted as:

$$N = (V, A, lo, up, co, start, fin) \tag{3}$$

where: V - is the vertex set, $V = F \cup S \cup \{start, fin\}$. We add two new vertices: *start* and *fin*; A - the set of edges of the graph; *lo* and *up* correspond to the lower and upper bound, while *co* is a set of functions that associates a real non-negative value to each edge;

The set of edges A and the functions lo, up, co are defined as following, where $|S|$ denotes the number of sites:

$$\forall f_i \in F, a_i = (start, f_i) \in A; lo(a_i) = 0; up(a_i) = df_i \times |S|; co(a_i) = 0; \tag{4}$$

$$\forall s_j \in S, a_j = (s_j, fin) \in A; lo(a_j) = 0; up(a_j) = ds_j; co(a_j) = 0; \tag{5}$$

$$\begin{cases} \forall (f_i, s_j, c_{ij}) \in P, a_{ij} = (f_i, s_j) \in A; \\ lo(a_{ij}) = up(a_{ij}) = df_i; co(a_{ij}) = \dfrac{c_{ij}}{df_i}; \end{cases} \tag{6}$$

A flow in the above transport network N is a real function: $fl : A \longrightarrow \Re$ having the following properties:

(1) Capacity Restrictions: $fl(a) = 0$ or $lo(a) \leq fl(a) \leq up(a), \forall a \in A$.
(2) Flow Conservation $\forall v \in V - \{start, fin\}$:

$$\sum_{\substack{u \in V, \\ (u,v) \in A}} fl(u, v) = \sum_{\substack{u \in V, \\ (v,u) \in A}} fl(v, u) \text{ or } \sum_{\substack{a=(u,v) \in A, \\ u \in V}} fl(a) = \sum_{\substack{a=(v,u), \\ u \in V}} fl(a)$$

The *value of the flow* can be computed by: $\sum_{s \in S} fl(s, fin)$. Given a flow in the network N, we can determine a cost given according to the following formula:

$$cost(fl) = \sum_{a \in A} fl(a) \times co(a).$$

where $co(a)$ is the value associated to each edge, introduced above and computed according to (4, 5, 6). A flow has a maximum cost if there is no other flow with a higher cost. Conditions from (6) state that storing a fragment f on site s is done on the entire fragment or not at all. Equation (5) state that storing fragments in a site cannot exceed the available storage space on that site. Equation (4) state that the number of fragment replicas is unbounded. The flow cost is only influenced by the values of the *co* function in (6) and its value represents the amount of cost reduction for data transfers.

Finding the allocation schema (with replication) that maximizes the data transfer cost reduction can be solved in the above conditions by finding the maximum compatible cost flow in the transport network.

This transport problem is a special one due to its capacity restrictions and the maximum cost requirement, but not the maximum flow. We elaborate a backtrack type algorithm which determines for every site s_i a set of fragments that can be replicated on that site. From all the constructed sets we only keep the one with the maximum cost. The main issue of the backtracking algorithm is that it performs an exhaustive search of the solution space. The explored space grows proportionally with the product of the number of fragments and sites and the required time to solution grows and becomes unrealistic for an online system. As a solution to this issue we elaborate an approximate algorithm based on a greedy approach to find the maximum cost flow. The previous algorithm is simplified by considering only one set R in step 2. Candidate replication fragments will be allocated to a site in cost descending order - as long as there is available space. This approach reduces algorithm complexity while still allowing a close approximation of the solution. We thought our proposal as a module in a database system, that runs quasi-continuously and provides replication hints whenever these exist and are possible. As a consequence the algorithm should be as fast as possible and with minimal impact on the database.

```
Algorithm Max Cost Flow Greedy:
INPUT:n (number of fragments);  df (n dimensional array - fragments’ dimension);
      m (number of sites);  ds (m dim. array - the available space on sites);
      c (m x n dim. array - transfer cost of fragment F[i] to size S[j];
1.INIT.
    FOR j=1,...,m
      fragm[j] = EMPTY SET  (index of the fragment replicated on site j)
2.FOR every site j=1,...,m construct array fragm[j]
  2.1 SumDimF:=0; R = empty set;
      sort descending fragment transport costs: c[j][1],...,c[j][n]
      LET PF[1],...,PF[n] be the fragments in descending costs order
      FOR i=1,...,n
        --site j has enough space to store fragment PF[i]
        IF (SumDimF+df[PF[i]]<=ds[j] and c[j][PF[i]]>0)
        Add PF[i] to R;
        update SumDimF;
        store R in fragm[j];
OUTPUT: fragm[j], j=1,...,m  --fragments to replicate in site j
```

5 Experimental Results

In order to evaluate the efficiency of the proposed solutions, we run a battery of simulations and tests. For assessing the generality of our model we randomly generate a set of database configurations. To test the proposed Induced Fragment Replication (IFR) we generate different sets of large distributed databases. The

Table 1. Test distributed DB configurations

DDB	Frag	Sites	Queries	Max reps
DDB1	20	5	10000	100
DDB2	50	5	10000	100
DDB3	100	5	10000	100
DDB4	200	5	10000	100
DDB5	300	5	10000	100
DDB6	20	10	10000	100
DDB7	50	10	10000	100
DDB8	100	10	10000	100
DDB9	200	10	10000	100
DDB10	300	10	10000	100
DDB11	20	20	10000	100
DDB12	50	20	10000	100
DDB13	100	20	10000	100
DDB14	200	20	10000	100
DDB15	300	20	10000	100

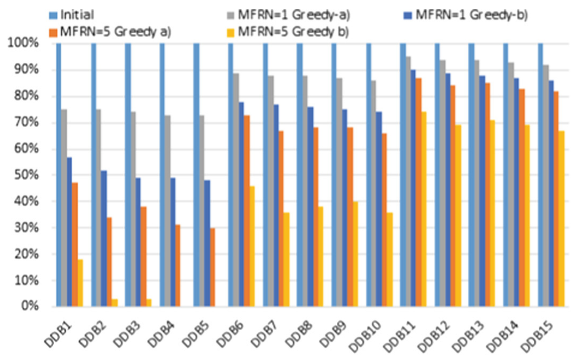

Fig. 4. Cost Improvements Percents for MFRN = 1 and MFRN = 5

synthetic experiments were preferred due to the lack of large and statistically complete and consistent real databases. We choose to generate statistically database configurations as we only need to process the meta-information from the database and not the actual data. We first generated an initial database state by averaging the evaluation costs over a number of uniformly sampled generated distribution configurations. Then we generate fifteen small to large sample database configurations drawn from the same distribution (see [11] about the configuration generator, test data and results) (Fig. 4).

The fifteen distributed database configurations are presented in Table 1.

In the following we present the analysis of the tests' results. In order to asses the improvements we measure the network transfer before and after applying the induced fragmentation on a series of test databases. We consider the percent of data transfer cost needed in query processing after applying the IFR solution compared to the transfer cost in the initial database as the measure of query optimization. We test the replication problem for the next cases: (a) the available space is equal with the space occupied by the fragments; and (b) the available space is 2 * space occupied by the fragments. Table 2 presents the percents and execution times in seconds for IFR problem when (Maximum Fragment Replication Number - the maximal number of generated fragment replicas) MFRN=1 and MFRN=5 and the available free space corresponds to above space constraints (a) and (b).

The proposed network flow problem is a bit uncommon as it needs a flow of maximum cost (regardless of the value of the flow). There is no solver, to our knowledge, for this problem formulation and thus we implemented a backtracking solution to solve the flow problem and then we proposed a faster Greedy approximation algorithm for the same problem.

In Table 2 we show the transport cost expressed as a percentage of the original database (before applying induced fragmentation). We also present the execution times in seconds for each algorithm variant. The backtracking variant has an

Table 2. Costs and exec times for MFRN = 1 and MFRN = 5, cases (*a*) and (*b*)

	MFRN=1								MFRN=5							
	case (a)				case (b)				case (a)				case (b)			
	Greedy		Backtrack		Greedy		Backtrack		Greedy		Backtrack		Greedy		Backtrack	
DDB	cost (%)	time (s)	cost (%)	time (s)	cost (%)	time (s)	cost (%)	time (s)	cost (%)	time (s)	cost (%)	time (s)	cost (%)	time (s)	cost (%)	time (s)
DDB1	75	0.2	74	0.97	57	0.22	55	2.6	47	0.22	46	1.01	18	0.23	17	4.51
DDB2	75	0.21			52	0.27			34	0.29			3	0.28		
DDB3	74	0.31			49	0.3			38	0.31			3	0.26		
DDB4	73	0.39			49	0.33			31	0.34			0	0.43		
DDB5	73	0.46			48	0.51			30	0.51			0	0.48		
DDB6	89	0.27	88	0.1	78	0.33	77	0.1	73	0.21	72	1.31	46	0.25	45	9.65
DDB7	88	0.51			77	0.31			67	0.33			36	0.33		
DDB8	88	0.43			76	0.43			68	0.33			38	0.36		
DDB9	87	0.53			75	0.63			68	0.56			40	0.59		
DDB10	86	0.7			74	0.74			66	0.79			36	0.7		
DDB11	95	0.31	95	0.1	90	0.32	89	0.1	87	0.27	87	1.1	74	0.29		
DDB12	94	0.39	94	225	89	0.42			84	0.39			69	0.38		
DDB13	94	0.51			88	0.46			85	0.49			71	0.48		
DDB14	93	0.83			87	0.93			83	0.82			69	0.82		
DDB15	92	1.24			86	1.13			82	1.04			67	1.19		

exact solution but the search space explodes exponentially with the product of the number of fragments, sites and queries and thus its execution times explode exponentially with this product (search space). In Table 2 we left empty the cells where the backtracking solution failed to give a solution within the time required for a Mathematical solver to solve the equivalent linear programming problem.

The proposed Greedy solution has an almost constant execution time with a ratio of 6:1 between the fastest and slowest solution. As execution time this is more than appropriate for a system where our module is run quasi-continuously and produces fragmentation hints.

The cost penalty obtained by applying the proposed approximation Greedy solution is around 1.75% higher compared to the exact backtracking solution. However the execution time for the Greedy approach compared to the exact solutions is in the order of 550 times less. The Greedy solution average running time is around 0.5 s with the largest execution time being 1.24 s.

6 Conclusions and Future Work

In this paper we provide a solution for query response time improvement modeled as a maximal compatible flow cost in transport networks. We perform online perpetual data replication within the original space constraints of the database. The major contribution is the Greedy algorithm that solves the transport flow problem in a fraction of the time needed to a classical solver or algorithm with an approximation penalty cost under 2% making this algorithm suitable to online execution within the database and as a replacement to the classical solvers.

We only considered so far the transport cost as the argument driving data replication and allocation in order to improve query response time. While it solves a complex problem this model is in many cases too simplistic. As future work we would like to extend our model to cases where data allocation is driven by more parameters (CPU, storage, network capacities, etc.) or to non-relational cloud databases where principles are different.

References

1. Apers, P.M.G.: Data allocation in distributed database systems. ACM T Datab. Syst. **13**(3), 263–304 (1988). Applied Mathematical Programming. Addison-Wesley (1977)
2. Cheng, C.H., Lee, W.K., Wong, K.F.: A genetic algorithm-based clustering approach for database partitioning. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **32**(3), 215–230 (2002)
3. Dokeroglu, T., Bayır, M.A., Cosar, A.: Integer linear programming solution for the multiple query optimization problem. In: Czachórski, T., Gelenbe, E., Lent, R. (eds.) Information Sciences and Systems 2014, pp. 51–60. Springer, Cham (2014). doi:10.1007/978-3-319-09465-6_6
4. Graham, J., Foss, J.A.: Efficient allocation in distributed object oriented databases. In: Proceedings of 16th International Conference on Parallel and Distributed Computing Systems (ISCA), pp. 471–412 (2003)
5. Huang, Y., Chen, J.: Fragment allocation in distributed database design. J. Inf. Sci. Eng. **17**, 491–506 (2001)
6. Lin, X., Orlowska, M.: An integer linear programming approach to data allocation with the minimum total communication cost in distributed database systems. Inf. Sci. **85**, 1–10 (1995)
7. Menon, S.: Allocating fragments in distributed databases. IEEE Trans. Parallel Distrib. **16**(7), 577–585 (2005)
8. Morffi, A.R., et al.: A reinforcement learning solution for allocating replicated fragments in a distributed database. Comput. Sist. **11**(2), 117–128 (2007)
9. Ozsu, M.T., Valduriez, P.: Principles of Distributed Database Systems. Springer, Heidelberg (2011)
10. Tambulea, L., Darabant, A.S., Varga, V.: Data transfer optimization in distributed database query processing. Studia Univ Babes Bolyai, Informatica **LIX**(1), 71–82 (2014)
11. Tambulea, L., Darabant, A. S., Varga, V.: Query Evaluation Optimization in a Distributed Database using Data Reorganization (2015). http://www.cs.ubbcluj.ro/~ivarga/ddbpaper
12. Virk, R.S., Singh, D.G.: Optimizing access strategies for a distributed database using genetic fragmentation. Int. J. Comput. Sci. Netw. Secur. **11**(6), 180–183 (2011)
13. Wiese, L.: Clustering-based fragmentation and data replication for flexible query answering in distributed databases. Int. J. Cloud Comput. **3**(1), 3–18 (2014)