Pakistan Journal of Information and Technology 2 (3): 231-239, 2003
ISSN 1682-6027

Heuristic Approach to Dynamic Data Allocation in Distributed Database Systems

[1]Tolga Ulus and [2]Mithat Uysal
[1]Management Information Systems Department,
Bo–aziçi University, Bebek 34342 ¤stanbul, Turkey
[2]Computer Engineering Department, Do–uÕ University, Kad¥köy 34722 ¤stanbul, Turkey

Abstract: In this paper, a new dynamic data allocation algorithm for non-replicated distributed database systems (DDS), namely the threshold algorithm, is proposed. The threshold algorithm reallocates data with respect to changing data access patterns. The algorithm is analyzed for a fragment using simulation. The threshold algorithm is especially suitable for a DDS where data access pattern changes dynamically.

Key words: Distributed databases, fragmentation, data allocation, simulation

Introduction

Developments in database and networking technologies in the past few decades led to advances in distributed database systems. A DDS is a collection of sites connected by a communication network, in which each site is a database system in its own right, but the sites have agreed to work together, so that a user at any site can access data anywhere in the network exactly as if the data were all stored at the user's own site (Özsu and Valduriez, 1991).

The primary concern of a DDS is to design the fragmentation and allocation of the underlying database. Fragmentation unit can be a file where allocation issue becomes the file allocation problem. File allocation problem is studied extensively in the literature, started by Chu (Chu, 1969) and continued for non-replicated and replicated models (Morgan and Levin, 1977; Azoulay-Schwartz and Kraus, 2002). Some studies considered dynamic file allocation (Wah, 1979; Smith, 1981).

Data allocation problem was introduced when Eswaran (Eswaran, 1974) first proposed the data fragmentation. Studies on vertical fragmentation (Navathe *et al*., 1984; Ceri *et al*., 1989); horizontal fragmentation (Ceri *et al*., 1983) and mixed fragmentation (Sacco, 1986; Zhang and Orlowska, 1994; Cheng *et al*., 2002) were conducted. The allocation of the fragments is also studied extensively (So *et al*., 1999; Bakker, 2000; Ahmad *et al*., 2002 and Chang, 2002).

In these studies, data allocation has been proposed prior to the design of a database depending on some static data access patterns and/or static query patterns. In a static environment where the access probabilities of nodes to the fragments never change, a static allocation of fragments provides the best solution. However, in a dynamic environment where these probabilities change over time, the static allocation solution would degrade the database performance. Initial studies on dynamic data allocation give a framework for data redistribution (Wilson and Navathe, 1986) and demonstrate how to perform the redistribution process in

minimum possible time (Rivera-Vega *et al*., 1990). In (Brunstorm *et al*., 1995); a dynamic data allocation algorithm for non-replicated database systems is proposed, but no modeling is done to analyze the algorithm. Instead, the paper focused on load balancing issue.

This paper proposes a new dynamic data allocation algorithm for non-replicated distributed databases and analyzes the algorithm using simulation. In our study, horizontal, vertical or mixed fragmentation can be used. Allocation unit can even be as small as a record or an attribute.

The rest of this paper is organized as follows. Firstly, the optimal dynamic data allocation algorithm given in (Brunstorm *et al*., 1995) is revisited and its disadvantages are discussed. After that, a new algorithm (namely the threshold algorithm), which overcomes the disadvantages of the optimal algorithm, is proposed for dynamic data allocation in distributed databases. Next, the behavior of a fragment, in reaction to a change in access probabilities or to a change in threshold value, is investigated using simulation. Finally, concluding remarks are given.

Optimal algorithm

In distributed database systems, the performance increases when the fragments are stored at the nodes from which they are most frequently accessed. The problem is to find this particular node for each fragment. Counting the accesses of each node to a fragment offers a practical solution. Having the highest access value for a particular fragment, a node could be the primary candidate to store the fragment.

An m by n access counter matrix S, where m denotes the number of fragments and n denotes the number of nodes, is shown below.

$$S' \begin{bmatrix} s_{11} & s_{12} & .. & s_{1n} \\ s_{21} & s_{22} & .. & s_{2n} \\ : & : & :: & : \\ s_{m1} & s_{m2} & .. & s_{mn} \end{bmatrix}$$

Every element s_{ij} of S, where $s_{ij} 0 Z^{+} c \{0\}$ (i.e. non-negative integers), shows the number of accesses to fragment I by node j. A row of S shows the access counts of all nodes to a particular fragment, whereas a column of S shows the access counts of all fragments for a particular node. S is updated after each access to the fragment. To satisfy the minimum response time constraint, each fragment should be stored in the node with the highest access count in its corresponding row. In other words, if $s_{ix} > s_{iy}$ for all y = 1,..,n, then the fragment I should be store in node x.

For example let the following matrix be the access counter matrix, S, of a DDS with three nodes and three fragments.

$$\begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{ccc} x & y & z \end{array} \\ \begin{bmatrix} 3 & 6 & 2 \\ 5 & 3 & 1 \\ 1 & 4 & 2 \end{bmatrix} \end{array}$$

In the example, A, B, C denote fragments and x, y, z denote nodes. Fragment A is accessed 3, 6 and 2 times by nodes x, y and z, respectively. For fragments B and C, the access counts are 5, 3, 1 and 1, 4, 2, respectively. According to S, fragment A and C should be stored in node y; and fragment B should be stored in node x.

The storage node of S is an issue of the algorithm. To store it in a central node creates an extra network traffic overhead. It also leads to a reliability problem in case of central node failure. The best solution would be to decompose the matrix into rows and store each row together with its associated fragment in the same node. In this way, whenever the fragment migrates, its associated counters migrate as well. Fig. 1 shows fragment I with its associated counters, s_0 through s_n.

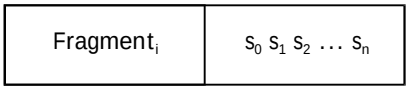

Fig. 1: Any fragment I in optimal algorithm

Initially, all fragments are distributed to the nodes according to any method. Afterwards, any node j, runs the optimal algorithm given in Fig. 2 for every fragment I, that it stores.

Step 1. For each (locally) stored fragment, initialize the access counter rows to zero. (S_{ik} = 0 were k = 1,..,n)
Step 2. Process an access request for the stored fragment
Step 3. Increase the corresponding access counter of the accessing node for the stored fragment. (If node x accesses fragment I, set $s_{ix} = s_{ix} + 1$)
Step 4. If the accessing node is the current owner, go to step 2. (i.e.. Local access, otherwise it is a remote access)
Step 5. If the counter of a remote node is greater than the counter of the current owner node, transfer the ownership of the fragment together with the access counter array to the remote node. (i.e. fragment migrates) (if node x accesses fragment I and $s_{ix} > s_{ij}$, send fragment I to node x)
Step 6. Go to step 2.

Fig. 2: Optimal algorithm

There are two inherent properties introduced by the optimal algorithm. First one is the ownership property, that is, for each fragment; the node with highest access counter value is the current owner node of the fragment, in which case the fragment is stored in this node.

The second one, namely migration property, dictates that for any fragment the ownership is transferred to a new node, if the access counter value of the new node exceeds the access counter value of current owner node. In this case, this particular fragment migrates and is stored in this new owner node. In other words, the owner node of the fragment changes.

An advantage of the optimal algorithm is the central node independence. That is, since each node runs the algorithm autonomously, there is no central node dependence. Every node is of equal importance. Whenever one node crashes, the algorithm may continue its operation without the fragments stored in the crashed node.

There are two drawbacks associated with the optimal algorithm. First one is the potential storage problem. As the fragment size decreases and/or the number of nodes increases, the size of access counter matrix increases, which in turn results in extra storage space need for the access counter matrix. For instance, if the fragment size is one record and the number of nodes is 500, then for each record an array of 500 access counter values should be stored. In some cases, this access counter array size may exceed the record size.

The second drawback is the scaling problem for the data type that stores the access counter values. Since access counter values are continuously increasing, this problem may result in anomalies. For example, if one byte is chosen to store the counter values, then a value greater than 255 cannot be stored in this data type.

A potential timing problem, which may cause back and forth migration of a fragment, deserves explanation. Think of a case where there are two nodes denoted by Y and Z. Suppose at an instant, Y has the highest access counter for a fragment whereas Z's counter is one less than Y's. Consider Z performs two successive accesses to the fragment, causing a transfer of the fragment from Y to Z. Later on, Y performs two successive accesses to the fragment, causing again the transfer of the fragment from Z to Y. If these successive two accesses of Y and Z continue in turns, the fragment will migrate back and forth between Y and Z after every two accesses. This can also be generalized to multiple nodes where fragments migrate in a circular fashion.

The threshold algorithm

In some cases, due to extra storage space need, it could be very costly to use the optimal algorithm in its original form. For a less costly algorithm, the solution is to decrease the need for extra storage space. The heuristic threshold algorithm in this paper serves this purpose.

Let the number of nodes be n and let x_s denote the access probability of a node to a particular fragment. Suppose the fragment is stored in this particular node (i.e. it is the owner node). For the sake of simplicity, let x_d denote the access probability of all the other nodes to this particular fragment. The owner does local access, whereas the remaining nodes do remote access to the fragment.

The probability that the owner node does not access the fragment is $(n - 1)x_d$. The probability that the owner node does not perform two successive accesses is $[(n - 1)x_d]^2$. Similarly, the probability that the owner node does not perform m successive accesses is $[(n - 1)x_d]^m$. Therefore; the probability that the owner node performs at least one access of m successive accesses is $1-[(n - 1)x_d]^m$.

Table 1 shows the probabilities that the owner node performs at least one access out of m successive accesses, where x_s ranges from 0.1 through 0.9 and where m is 5, 10, 25, 50 and 100. The values in the table are truncated to five decimal digits.

Table 1: The probability that at least one local access occurs in m accesses

x_s	m=5	m=10	m=25	m=50	m=100
0,1	0,40951	0,65132	0,92821	0,99485	0,99997
0,2	0,67232	0,89263	0,99622	0,99999	1,00000
0,3	0,83193	0,97175	0,99987	1,00000	1,00000
0,4	0,92224	0,99395	1,00000	1,00000	1,00000
0,5	0,96875	0,99902	1,00000	1,00000	1,00000
0,6	0,98976	0,99990	1,00000	1,00000	1,00000
0,7	0,99757	0,99999	1,00000	1,00000	1,00000
0,8	0,99968	1,00000	1,00000	1,00000	1,00000
0,9	0,99999	1,00000	1,00000	1,00000	1,00000

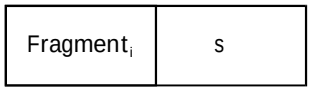

Fig. 3: Any fragment I in threshold algorithm

According to the table, the probability that the owner node with the access probability of 0.1 performs at least one access of ten successive accesses is 0.65132. It is trivial from the table that as the access probability of owner node increases, so as the probability that at least one local access occurs in m accesses.

Applying the same idea, a new threshold based algorithm (or threshold algorithm) can be proposed. In threshold algorithm, only one counter per fragment is stored. Fig. 3 shows fragment I together with its counter. Comparing it to the optimal algorithm, this radically decreases the extra amount of storage space to just one value compared to an array of values in the optimal algorithm.

In the threshold algorithm, the initial value of the counter is zero. The counter value is increased by one for each remote access to the fragment. It is reset to zero for a local access. In other words, the counter always shows the number of successive remote accesses. Whenever the counter exceeds a predetermined threshold value, the ownership of the fragment is transferred to another node.

At this point, the critical question is which node will be the fragment's new owner. The algorithm gives very little information about the past accesses to the fragment. In fact, throughout the entire access history only the last node that accessed the fragment is known. So, there are two strategies to select the new owner. Either it is chosen randomly, or the last accessing node is chosen. In the former, the randomly chosen node could be one that has never accessed the fragment before. So picking the latter strategy is heuristically more reasonable.

Initially, all fragments are distributed to the nodes according to any method. A threshold value t is chosen. Afterwards, any node j, runs the threshold algorithm given in Fig. 4 for every fragment I, that it stores.

Threshold algorithm overcomes the volley of a fragment between two nodes provided that a threshold value greater than one is chosen. The algorithm guarantees the stay of the fragment for at least (t+1) accesses in the new node after a migration. In other words, it delays the migration of the fragment from any node for at least (t+1) accesses.

Step 1. For each (locally) stored fragment, initialize the counter values to zero. (Set $s_i = 0$ for every stored fragment I)
Step 2. Process an access request for the stored fragment.
Step 3. If it is a local access, reset the counter of the corresponding fragment to 0 (If node j accesses fragment I, set $s_i = 0$). Go to step 2.
Step 4. If it is a remote access, increase the counter of the corresponding fragment by one. (If fragment I is accessed remotely, set $s_i = s_i + 1$)
Step 5. If the counter of the fragment is greater than the threshold value, reset its counter to zero and transfer the fragment to the remote node. (If, $s_i > t$, set $s_i = 0$ and send the fragment to remote node)
Step 6. Go to step 2.

Fig. 4: Threshold algorithm

An important point in the algorithm is the choice of threshold value. This value will directly affect the mobility of the fragments. It is trivial that as the threshold value increases, the fragment will tend to stay more at a node; and as the threshold value decreases, the fragment will tend to visit more nodes.

Another point in the algorithm is the distribution of the access probabilities. If the access probabilities of all nodes for a particular fragment are equal, the fragment will visit all the nodes. The same applies for two nodes when there are two highest equal access probabilities.

Simulation results

In the simulation, it is assumed that there are n nodes; x_s is the access probability of the owner node; x_d is the access probability of the other nodes; O_s is the probability that the fragment is in owner node and O_d is the total probability that the fragment is in the other nodes.

Since, $O_s + O_d = 1$, investigating only O_s is sufficient. The following formula shows the relation between n, x_s and x_d.

$$x_s + (n-1)\, x_d = 1$$

Now, let us find how a change in the access probabilities and the threshold value effect the probability that the fragment is in any node.

Change in access probability

When n is held constant, x_s and x_d are inversely proportional. So, it is sufficient to investigate only the change in x_s of O_s.

Fig. 5 shows the behaviour of O_s as a function of x_s in a five-node system. Fig. 5 is drawn for three different threshold values, 0, 3 and 10.

For the threshold of 0, O_s is a linear function of x_s with a slope of 1. This means that when the threshold is 0, the access probability of a node directly gives the probability that the fragment is in the corresponding node.

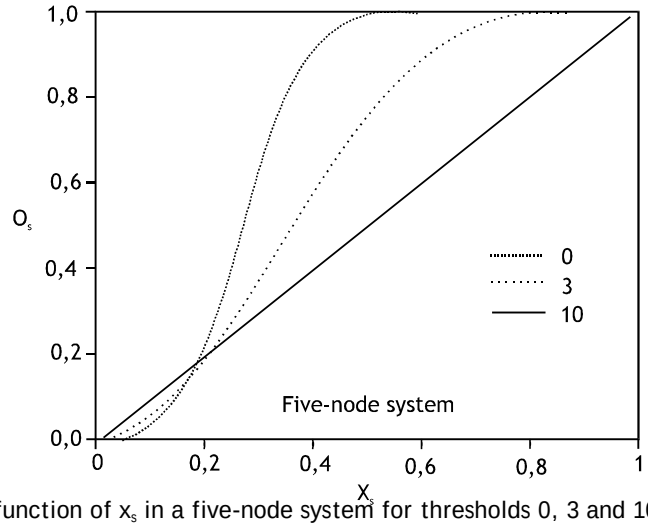

Fig. 5: O_s as a function of x_s in a five-node system for thresholds 0, 3 and 10

For threshold values of 3 and 10, notice the change in steepness of the curve.

Change in threshold value

Threshold t can take only non-negative integer values.

Fig. 6 shows the behaviour of O_s as a function of t in a five-node system. Fig. 6 is drawn for five different access probabilities x_s of 0.28, 0.24, 0.2, 0.16 and 0.12.

For 0.28 and 0.24, O_s converges to one. This is because $x_s > x_d$. Noticing the change in steepness of two curves, it converges faster for greater access probabilities.

For 0.2, O_s is constant at 0.2. This is because $x_s = x_d$. In this case, the access probability of a node directly gives the steady-state probability that the fragment is in the corresponding node.

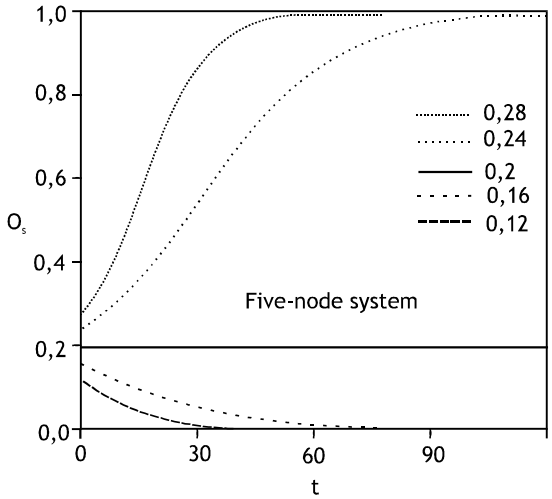

Fig. 6: O_s as a function of t in a five-node system for x_s values of 0.28, 0.24, 0.2, 0.16 and 0.12

For 0.16 and 0.12, O_s converges to zero. This is because $x_s \leq x_d$. Noticing the change in steepness of two curves, it converges faster for smaller access probabilities.

In this paper, a new dynamic data allocation algorithm, namely threshold algorithm, for non-replicated DSSs is introduced. In the threshold algorithm, the fragments, previously distributed over a DDS, are continuously reallocated according to the changing data access patterns.

The behavior of a fragment, in reaction to a change in access probabilities or to a change in threshold value, is investigated using simulation. It is shown that the fragment tends to stay at the node with higher access probability. As the access probability of the node increases, the tendency to remain at this node also increases. It is also shown that as the threshold value increases, the fragment will tend to stay more at the node with higher access probability.

Threshold algorithm can be used for dynamic data allocation to enhance the performance of non-replicated DDSs. For further research, the algorithm can be extended to use on the replicated DSSs as in (Wolfson and Jajodia, 1997; Sistla *et al*., 1998).

References

Ahmad, I., K. Karlapalem, Y. K. Kwok and S. K. So., 2002. Evolutionary Algorithms for Allocating Data in Distributed Database Systems, Distributed and Parallel Databases, 11: 5-32.

Azoulay-Schwartz, R. and S. Kraus, 2002. Negotiation on Data Allocation in Multi-Agent Environments, Autonomous Agents and Multi-Agent Systems, 5: 123-172.

Bakker, J.A., 2000. Semantic Partitioning as a Basis for Parallel I/O in Database Management Systems", Parallel Computing, 26: 1491-1513.

Brunstrom, A., S.T. Leutenegger and R. Simha, 1995. Experimental Evaluation of Dynamic Data Allocation Strategies in a Distributed Database With Changing Workloads, in Proceedings of the 1995 International Conference on Information and Knowledge Management, Baltimore, MD, USA, pp: 395-402.

Ceri, S., S.B. Navathe and G. Wiederhold, 1983. Distribution Design of Logical Database Schemas, IEEE Transactions on Software Engineering, 9:487-503.

Ceri, S., B. Pernici and G. Wiederhold, 1989. Optimization Problems and Solution Methods in the Design of Data Distribution, Information Systems, 14: 261-272.

Chang, C.T., 2002. Optimization Approach for Data Allocation in Multidisk Database, European J. Operational Res., 143: 210-217.

Cheng, C.H., W.K. Lee and K.F. Wong, 2002. A Genetic Algorithm-Based Clustering Approach for Database Partitioning, IEEE Transactions on Systems Man and Cybernetics Part C-Applications and Reviews, 32: 215-230.

Chu, W.W., 1969. Optimal File Allocation in a Multiple Computer System, IEEE Transactions on Computers, C-18: 885-889.

Eswaran, K.P., 1974. Placement of Records in a File and File Allocation in a Computer Network, in Proceedings of IFIP Congress on Information Processing, Stockholm, Sweden, pp: 304-307.

Morgan, H.L. and K.D. Levin, 1977. Optimal Program and Data Locations in Computer Networks, Communications of ACM., 20: 315-321.

Navathe, S.B., S. Ceri, G. Wiederhold and J. Dou, 1984. Vertical Partitioning Algorithms for Database Design, ACM Transaction on Database Systems, 9: 680-710.

Özsu, T. and P. Valduriez, 1991. Principles of Distributed Database Systems. Prentice-Hall Book Co., Englewood Cliff, USA.

Rivera-Vega, P.I., R. Varadarajan and S.B. Navathe, 1990. Scheduling Data Redistribution in Distributed Databases, in IEEE Proceedings of the Sixth International Conference on Data Engineering, pp: 166-173.

Sacco, G., 1986. Fragmentation: A Technique for Efficient Query Processing, ACM Transaction on Database Systems, 11: 113-133.

Sistla, A.P., O. Wolfson and Y. Huang, 1998. Minimization of Communication Cost Through Caching in Mobile Environments, IEEE Transactions on Parallel Distributed Systems, 9: 378-390.

Smith, A.J., 1981. Long-term File Migration: Development and Evaluation of Algorithms, Communications of ACM., 24: 512-532.

So, S.K., I. Ahmad and K. Karlapalem, 1999. Response Time Driven Multimedia Data Objects Allocation for Browsing Documents in Distributed Environments, IEEE Transactions on Knowledge and Data Engineering, 11: 386-405.

Wah, B.W., 1979. Data Management in Distributed Systems and Distributed Data Bases. Ph.D. Dissertation, University of California, Berkeley, CA, USA.

Wilson, B. and S.B. Navathe, 1986. An Analytical Framework for the Redesign of Distributed Databases, in Proceedings of the 6th Advanced Database Symposium, Tokyo, Japan, pp: 77-83.

Wolfson, O. and S. Jajodia, 1997. An Adaptive Data Replication Algorithm, ACM Transaction on Database Systems., 22: 255-314.

Zhang, Y. and M.E. Orlowska, 1994. On Fragmentation Approaches for Distributed Database Design, Information Sci., 1: 117-132.