

DATA FRAGMENTATION AND ALLOCATION FOR PC-BASED DISTRIBUTED DATABASE DESIGN*

Minyoung Ra, Yang-Sun Park

Computer Science Division
Department of Mathematics
Korea Military Academy
Seoul, KOREA 139-799

ABSTRACT

As the performance of PCs is improved, the need for PC-based distributed database systems has been increased. Data fragmentation and allocation is one of the major design issues for distributed database systems. In this paper, several factors that should be considered in constructing a PC-based distributed database system have been investigated, and a methodology for partitioning and allocating data effectively over a network for PC-based distributed database design is proposed. This methodology is based on the mixed partitioning technique using a grid approach.

1. INTRODUCTION

Recently, the use of Database Systems(DBS) on Personal Computers(PC) has been increased. A database system, which stores and manages data to produce useful information, is a computer oriented system consisting of databases(DB), a database management system (DBMS), data languages, and database administrators (DBAs), [Elma 89]. In the past years, DBS was mainly used on mini computers and mainframes, but now it is widely used on PCs because the performance of PCs has been improved and the cost of PCs came down.

However, most of the DBSs on PCs are for a single-user and thus sharing of resources is not considered. To make up for this weakness PCs are connected into distributed database systems to share resources. Research on distributed database systems (DDBS) has been increased because DDBSs can improve reliability and availability and fit more naturally in the decentralized structures of many organizations [Ceri 84, Oszu 91]. Data fragmentation and allocation called data distribution is the basis for constructing distributed database systems. Since it is possible to construct distributed database systems on PCs, the research on data distribution for PC-based distributed database design should be emphasized. In this paper a distribution scheme for PC based distributed database systems is proposed by using a mixed fragmentation technique. Section 2 deals with some factors that should be considered for

data distribution. In Section 3, a methodology for generating distribution scheme by using a mixed partitioning technique is proposed. Section 4 gives the conclusion.

2. CONSIDERATIONS FOR DATA DISTRIBUTION

There are some factors that should be considered for effective data distribution in constructing PC-based distributed database systems.

2.1 Data Fragmentation

Data fragmentation (or partitioning) is the process that divides a logical object (relation) from the logical schema of the database to several physical objects (files) in a stored database [Nava 84]. There are basically two different ways in data fragmentation: vertical partitioning and horizontal partitioning. Vertical partitioning is the process of dividing attributes into groups. Previous work on vertical partitioning has used objective functions to perform partitioning [Ceri 88, Corn 87, Hamm 79, Nava 84]. Since in these approaches, binary partitioning technique should be applied recursively and objective functions and compliment algorithms such as SHIFT algorithm [Nava 84] are needed, we developed a graph theoretic algorithm that generates all meaningful vertical fragments in one iteration [Nava 89]. Horizontal partitioning is the process of dividing tuples in a relation into groups of tuples. In the most of the previous approaches, the problem is that there may be lots of horizontal partitions since at worst case a horizontal partition can be composed of only one tuple [Ceri 82, Cer 83b, Yu 85]. Because of this reason a new horizontal partitioning technique using predicate clustering is currently being studied to overcome this drawback [Ra 91].

Mixing the two types of partitioning has considered to yield mixed partitioning. The need for mixed partitioning arises in distributed databases because database users usually access data subsets which are both vertical and horizontal fragments of global relations [Elma 89]. The examples of the previous work on mixed partitioning are available in [Aper 88, Nava 90].

2.2 Data Allocation

*This research was supported in part by KOSEF (Korea Science & Engineering Foundation) Grant No. 923-1100-088-1.

3. SCHEME FOR DISTRIBUTION DESIGN

Based on the above considerations, we develop a scheme for PC-based distributed database design.

3.1 Problem Description

Suppose we want to connect databases in PCs using a LAN. There is a database server in the LAN, which supports the distributed database environment. Then the distribution design problem for a PC-based distributed database system on the LAN can be described as follows.

"PCs having data processing facility are interconnected using LANs to form a distributed database system. Data are distributed according to the logical schema of the databases. Using appropriate partitioning techniques relations are partitioned, and the results are allocated to the database server and PCs in order to minimize the total data transmission cost. If replication is needed, we allow replication of data. We, however, do not consider network topology, communication channel, and band width. These factors may be included in the later stage".

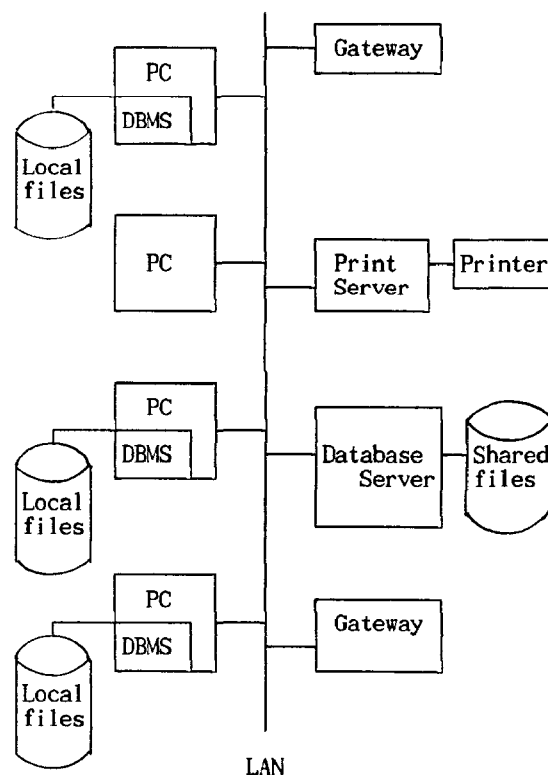


Fig. 1 PC-based distributed database environment

Figure 1 shows our research environment. As shown in Figure 1, we are investigating a server based on the client-server architecture, which processes data access requests from users, sends back the results to the users, maintains data

Data allocation is the process of mapping each logical fragment to one or more sites, and is considered as one of the major design issues for distributed databases [Cer 83a, Cer 83b, Corn 88]. Previous research in this area has been performed in two ways: data allocation by itself for distributed database systems, and extension of the pure data allocation problem by including the network topology and communication channels in the decision variables for distributed computing areas. This paper is concerned with pure data allocation in which the unit of allocation is the mixed fragment which comes from the mixed partitioning procedure in Section 3.2.1. The previous work in this area includes research on the models for data allocation [Cer 83b, Corn 88, Corn 89], and research on allocation heuristics [Aper 88, Ceri 88].

2.3 LAN

When we construct a distributed databases with PCs, the communication among them can usually be performed by using LANs. LANs are particular form of data communications, with hardware and software optimized to support the sharing of devices and information. They offer high degree of autonomy and versatility [Ceri 84, Patt 90]. The technologies related to PC LANs have progressed to replace mini computers, and making it possible to connect with large databases. The two important components comprising a LAN are a network operating system and a server. A network operating system is a system software that provides resource management for services on server machines, and provides the user and application software a window to the LAN environment. The following are key features that a NOS should support [Jord 90]: (1) hardware independence, (2) multiple server support, (3) multi user support, (4) network management, and (5) human engineering reflected user interface. The details are out of the present scope of this paper. Functions of a server are discussed in the next section.

2.4 Database Server

A server refers to a software application that offers a well-defined service to network users. A server application can be run on a special-purpose hardware or an ordinary PC. The most common types of servers are file servers, print servers, and communication servers [Cart 84, Jord 90]. A database server can be considered as a kind of file servers, which is divided into two categories: resource-shared architecture and client-server architecture. In resource-shared architecture, the users mainly work on PCs and send the results to the database server, or the users bring data from the database server and work within the PC. On the other hand, in client-server architecture, a database server processes database access requests from users and sends back the results to users. The criteria for evaluating database servers are (1) SQL-like query support, (2) various data types, (3) data compression, (4) record and database size limits, (5) recovery scheme and failure detection, (6) distributed processing support, and (7) installation and operation, etc. [Rein 89].

consistency, and protects data collision.

3.2 A Methodology for Distribution Design Scheme

The development of the scheme for distribution design scheme can be done in the following two steps.

- (1) mixed partitioning and grid optimization
- (2) allocation of mixed fragments

The next sections describes these steps in detail.

3.2.1 Mixed partitioning and grid optimization

A mixed partitioning methodology is a hybrid type of horizontal partitioning technique and vertical partitioning technique. Currently mixed partitioning has not been addressed in the literature. Today's methodology can produce mixed partitioning only in one of the following two ways: by performing horizontal partitioning followed by vertical partitioning or by performing vertical partitioning followed by horizontal partitioning. Obviously, this is not adequate since it potentially leads to different results and leaves out the possibility of combining fragments at a smaller granularity to produce more efficient data distribution. In this paper we adopt a uniform mixed partitioning methodology which generates optimal results called mixed fragments that are formed by merging grid cells to minimize the global transaction processing costs. Grid cells are created by applying independently vertical and horizontal partitioning algorithms to a relation. It should be noted that the mixed fragments cannot be otherwise produced by independent partitioning models. The efficient algorithms for horizontal and vertical partitioning are already presented respectively [Nava 89, Ra 90, Ra 91].

3.2.1.1 Mixed partitioning methodology

In this section we give an overview of the mixed partitioning methodology. It consists of the specification of inputs, vertical partitioning, horizontal partitioning, merging of grid cells, resulting in the generation of the fragmentation scheme.

(1) Specification of inputs: The following set of inputs need to be provided by the user in order to come up with the mixed fragmentation scheme.

(a) schema information : relations, attributes, cardinalities, attribute sizes, etc.

(b) transaction information : name, frequency, attribute usage, etc. The attribute usage matrix is a matrix containing transactions as rows and attributes as columns. Element $(i,j)=1$ if transaction i uses attribute j , else it is 0.

(c) distribution constraints : any predetermined partitions or fixed allocation of data.

(2) Vertical partitioning for grid

In this step all candidate vertical fragments are determined. We use a graphical algorithm [Nava 89] for generating all fragments in one iteration.

(3) Horizontal partitioning for grid

In this step, all candidate horizontal fragments are determined by using the same graphical technique. Note that the order of this and the above step can be interchanged. Figure 2 shows the transaction specifications for our example both vertical and horizontal partitioning, and an example of grid cells resulting from this specifications is shown in Figure 3.

(4) Grid optimization

In this step, cells are merged so as to minimize the global transaction processing cost. A cost model for evaluating the benefit of merging, and a heuristic greedy procedure to decide if and how the calls are merged, are the major issues that should be attacked.

(5) Generation of fragmentation scheme

The previous step gives us two types of merging options namely: merging of grid cells horizontally or vertically. In this step we map the two merging schemes to generate a set of mixed fragments giving rise to a fragmentation scheme.

Transactions	Attributes	Predicates	Number of accesses per time period
T1	a1,a5,a7	p1,p7	25
T2	a2,a3,a8,a9	p2,p7	50
T3	a4,a6,a10	p3,p7	25
T4	a2,a7,a8	p4,p8	35
T5	a1,a2,a3,a5,a7,a8,a9	p5,p8	25
T6	a1,a5	p6,p8	25
T7	a3,a9	p5,p8	25
T8	a3,a4,a6,a9,a10	p6,p8	15

Fig. 2 Transaction specifications

G 11	G 12	G 13
G 21	G 22	G 23
G 31	G 32	G 33
G 41	G 42	G 43
G 51	G 52	G 53

Fig. 3 Grid creation

3.2.1.2 Grid optimization

The step of grid optimization is the key point among the above steps. This is because the results of mixed partitioning called mixed fragments are generated after grid optimization. There, however, are no cost functions proposed so far, nor merging heuristics. Thus in this paper we propose a feasible solution for this problem.

(a) Transaction mapping

Transaction mapping is done by mapping attributes and the predicates of the transactions with the attributes and the predicates forming the

grid cells. Figure 4 shows the mapping of the transactions to grid cells. The transactions here do not mean 100% of expected transactions, but important transactions. Since the 80-20 rule applies to most practical situations, it is adequate to supply information regarding the 20% of the heavily used transactions which account for about 80% of the activity against the database. In this figure H_i represents the results of horizontal partitioning, and V_i represents the results of vertical partitioning. For example, transaction T4 accesses attributes a2, a7 and a8, and is based on predicates p4 and p8, whereas the grid cell G_{11} is formed of attributes a1, a5 and a7, and predicates p3, p4, p6 and p8. Hence transaction T4 access the cell G_{11} . Note that some of the grid cells are not accessed by the most important transactions, but may be accessed by other transactions. Based on this mapping information grid cells are merged to minimize total transaction processing cost.

(b) Cost model

Now we present the cost model that is the basis of grid optimization. To this end, we introduce the following notation.

- m : number of horizontal partitions
- n : number of vertical partitions
- N : number of transactions
- G_{ij} : grid cell determined by the i -th horizontal partition and the j -th vertical partition, where $1 \leq i \leq m$ and $1 \leq j \leq n$.
- R_{ij} : number of tuples in grid cell G_{ij}
- L_{ij} : total length of the attribute (in bytes) of a tuple of G_{ij}
- c_t : average cost of access of a tuple of the relation
- l_t : total length of a tuple in the given relation
- α : ratio of the cost of accessing a merged fragment to the cost of accessing its constituent grid cells
- C_{ij} : average cost of accessing grid cell G_{ij}

We assume a linear cost access model in that the cost of accessing a grid cell G_{ij} is given by $C_{ij} = R_{ij} * (L_{ij}/l_t) * c_t$. When a transaction needs to access attributes in two horizontal cells that are not merged, there is a need to perform 'matching' of keys in the data obtained from the two cells. This operation is a form of join and we refer to this cost as 'join cost'. Similarly, when a transaction needs to access tuples in two vertical cells that are not merged, there is a cost due to the need to perform union of the resultant tuples which we refer to as the 'union cost'. On the other hand, when a transaction needs data from only one of the constituent cells of a fragment, there is an additional cost due to the need to 'project' the attributes and 'select' the tuples of the grid cell from the fragment as needed by a transaction. Specifically, we need the following additional notation.

- $S(f)$: set of constituent grid cells of fragment f
- $F_k(f)$: frequency of accessing only fragment f by k -th transaction
- $C(f)$: average cost of accessing fragment f
- $F_k(f_1, f_2)$: frequency of accessing both the fragments f_1 and f_2 by the k -th transaction
- $J_k(f_1, f_2)$: join cost in processing fragments f_1 and f_2 for k -th transaction
- $U_k(f_1, f_2)$: union cost in processing fragments f_1 and f_2 for k -th transaction
- $P_k(f)$: projection cost in processing fragment f for k -th transaction
- $S_k(f)$: selection cost in processing fragment f for k -th transaction
- $TC_k(f_1, f_2)$: total cost or saving for k -th transaction by merging fragments (horizontal or vertical) f_1 and f_2

We use the term "fragment" to denote the result of merging one or more vertical and/or horizontal cells. The two vertical fragments have the same set of attributes (and hence same vertical grid cells) while two horizontal fragments have the same set of tuple ID's (and hence same horizontal grid cells). The average cost of accessing fragment f consisting of two or more grid cells is given by

$$C(f) = \alpha \sum_{G_{ij} \in S(f)} C_{ij}$$

We refer to the combined fragment obtained by merging two fragments f_1 and f_2 as $f_1 \cup f_2$. If V_1 and V_2 are two vertical fragments that can be merged horizontally, then

$$TC_k(V_1, V_2) = F_k(V_1 \cup V_2)(C(V_1 \cup V_2) + P_k(V_1 \cup V_2)) - (F_k(V_1)C(V_1) + F_k(V_2)C(V_2) + F_k(V_1, V_2)J_k(V_1, V_2))$$

Similarly if H_1 and H_2 are two horizontal fragments that can be merged vertically, then

$$TC_k(H_1, H_2) = F_k(H_1 \cup H_2)(C(H_1 \cup H_2) + S_k(H_1 \cup H_2)) - (F_k(H_1)C(H_1) + F_k(H_2)C(H_2) + F_k(H_1, H_2)U_k(H_1, H_2))$$

We note that it is beneficial to merge V_1 and V_2 if

$$\sum_{k=1}^N TC_k(V_1, V_2) < 0.$$

Similarly it is beneficial to merge H_1 and H_2 if

$$\sum_{k=1}^N TC_k(H_1, H_2) < 0.$$

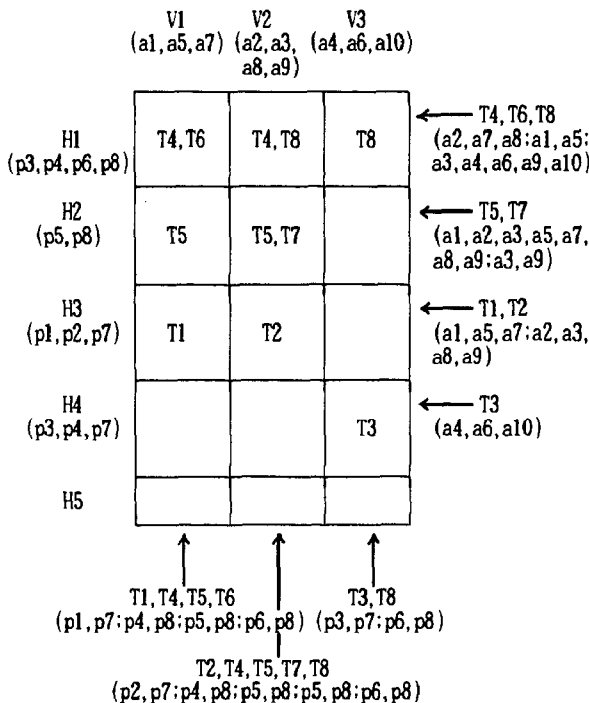


Fig. 4 Transaction mapping

There is a trade-off between all these costs. This trade-off determines an optimal merging of grid cells. It is possible to use an integer programming formulation that determines an optimal clustering of grid cells either horizontally or vertically. But the formulation will be difficult to solve and the solution procedure lacks intuition. For this reason, we use a heuristic procedure that performs successive merging based on the costs.

(c) Heuristic procedure for merging

Based on the cost functions we want to merge cells as much as possible in order to minimize the transaction processing cost as the given relation is concerned. We perform two kinds of merging, namely horizontal merging and vertical merging. Horizontal merging deals with merging of the cells in the same row of the grid.

In horizontal merging, the total number of ways of merging is $\sum_{i=1}^n {}^n C_i = 2^n - 1$ where ${}^n C_i$ represents combination selecting i form n and n represents the number of horizontal or vertical cells, because the sequence of attributes has no meaning in a relation. In our approach, however, we can minimize the possible ways of combinations of horizontal merging by using the ordered sequence of fragments generated in vertical partitioning. Thus, given n candidate horizontal fragments, a total of $(n-1) + (n-2) + \dots + 1 = n(n-1)/2$ merging possibilities are generated. This is because, in the linearly connected spanning tree in vertical partitioning, since a cut edge between two vertical fragments is a bridge that has the maximum affinity value among all connectable edges [Nava 89], we can say that a fragment is more closely related to contiguous fragments compared to noncontiguous fragments.

In vertical merging, cells in the same vertical column of the grid may be merged to produce larger fragments. Since the same graphical partitioning approach is used for horizontal partitioning, the total number of possible ways of vertical merging can be minimized in the same way as in the horizontal merging.

Thus in our heuristic procedure, only adjacent horizontal or vertical cells or fragments are considered for merging. We use an iterative greedy procedure in which we successively merge two adjacent fragments either horizontally or vertically until no more saving can be obtained by the merging process. The two adjacent horizontal or vertical fragments that are chosen for merging in each iteration are the pair that yield the maximum savings by merging them together instead of keeping them apart. The heuristic procedure is given as follows:

Procedure MergeCells :

{ Initially each cell by itself is a fragment }

Repeat

1. For each pair of adjacent horizontal or vertical fragments F_1 and F_2 , calculate $TC(F_1, F_2) = \sum_{k=1}^N TC_k(F_1, F_2)$, where $TC_k(F_1, F_2)$ is as defined before.
2. If there exist at least one pair F_1 and F_2 that has $TC(F_1, F_2) < 0$, then find the pair F_1 and F_2 that has the least value of $TC(F_1, F_2)$ and merge them into a

horizontal or vertical fragment.

Until No pair (F_1, F_2) exists with $TC(F_1, F_2) < 0$.

End MergeCells.

We note that in step 1, the cost calculation is necessary only for the fragments adjoining the merged fragment of the previous iteration. Thus the computation required in each iteration is small.

Example :

Figure 5 shows the costs of accessing each of the horizontal fragments H1, H2, H3, H4 and H5 (i.e. average cost of access of a tuple of the relation times the number of tuples in the grid cell).

They are 100, 150, 200, 75, and 125 (i.e. c_1, c_2, c_3, c_4 and c_5) respectively. The length of the attributes of each of the vertical fragments are 14, 20 and 16 (i.e. l_1, l_2 and l_3) respectively. The cells are denoted by G_{ij} , $1 \leq i \leq 5$ and $1 \leq j \leq 3$. Hence the cost of accessing a single grid cell G_{ij} is $C_{ij} = c_i * (l_j / \sum_{j=1}^3 l_j)$. Therefore the cost of accessing grid cell G_{22} is $C_{22} = 150 * (20 / (14+20+16)) = 60$.

Cost	Attributes length		
	14	20	16
100	G 11	G 12	G 13
150	G 21	G 22	G 23
200	G 31	G 32	G 33
75	G 41	G 42	G 43
125	G 51	G 52	G 53

$$C_{ij} = \text{cost of accessing the } i\text{th horizontal fragment} \times \frac{L(\text{attributes of the } j\text{th vertical fragment})}{L(\text{all attributes})}$$

i.e) $C_5 = 150 \times \frac{20}{14 + 20 + 16} = 60$

Fig. 5 Cost model for merging grid cells

In the following illustration we use the join cost as a variable to show how merging is dependent upon the join cost. For simplicity in notation we denote grid cell G_{11} as G_1 and grid cell G_{12} as G_2 , and cost of accessing them as C_1 and C_2 respectively. The fragment formed by merging cells G_1 and G_2 as G_{12} with the cost of accessing as C_{12} . The frequency of transactions accessing the cells G_1, G_2 and G_{12} is denoted F_{1k}, F_{2k} and F_{12k} respectively. The parameter α is assumed to be 1.2. The projection cost is assumed to be 10.

The cost without merging is:

$$\begin{aligned} \sum_{k=1}^8 F_{1k}C_1 &= F_{14} * C_1 + F_{16} * C_1 \\ &= 35 * 28 + 25 * 28 = 1680 \end{aligned}$$

$$\begin{aligned} \sum_{k=1}^8 F_{2k}C_2 &= F_{24} * C_2 + F_{28} * C_2 \\ &= 35 * 40 + 15 * 40 = 2000 \end{aligned}$$

$$\sum_{k=1}^8 F_{12k}J = F_{124}J = 35 * J$$

Where J is the cost of joining the grid cells G₁ and G₂ and the only transaction T₄ accesses the merged fragment with frequency 35.

The cost with the merged cells is:

$$\begin{aligned} \sum_{k=1}^8 F_{12k}C_{12} &= F_{124} * C_{12} + F_{126} * C_{12} \\ &\quad + F_{128} * C_{12} \\ &= 35 * 81.6 + 25 * 81.6 \\ &\quad + 15 * 81.6 \\ &= 6120 \end{aligned}$$

$$\begin{aligned} \sum_{k=1}^8 F_{1k}P_{1/12} + \sum_{k=1}^8 F_{2k}P_{2/12} &= F_{16} * P_{1/12} + F_{28} * P_{2/12} \\ &= 25 * 10 + 15 * 10 \\ &= 400 \end{aligned}$$

$$\begin{aligned} \text{Where } C_{12} &= \alpha * (C_1 + C_2) \\ &= 1.2 * (28 + 40) = 81.6 \end{aligned}$$

We shall merge the two cells if:
 $1680 + 2000 + 35 * J > 6120 + 400$
 i.e. if $35 * J > 2840$ or if $J > 81.11$

The objective of above example was to show that the two cells are merged on the basis of the join cost and the frequencies of the transactions accessing the cells. We call the results of the merging "mixed fragments". Figure 6 shows one feasible result of merging.

F1	F2	
F3		X1
F4	F5	X2
X3	X4	F6
X5	X6	X7

Fig. 6 Grid optimization

3.2.2 Allocation of mixed fragments

After merging grid cells, the allocation of the mixed fragments for a PC based distributed databases can easily be performed by using the transaction mapping information. Since the origin sites of each transaction are fixed, the fragments requested from only one site are allocated to the requesting sites. However, the fragments that are not accessed are allocated to a server for future use. For example, fragment F₄ in Figure 6 is allocated to site S₁ where transaction T₁ is originated, F₅ to S₂, and F₆ to S₃ (See Figure 7, we assume that transaction T₁, T₄, and T₇ be originated from site S₁, etc.). Non-accessed fragments namely X₁ to X₇ are all allocated to a database server.

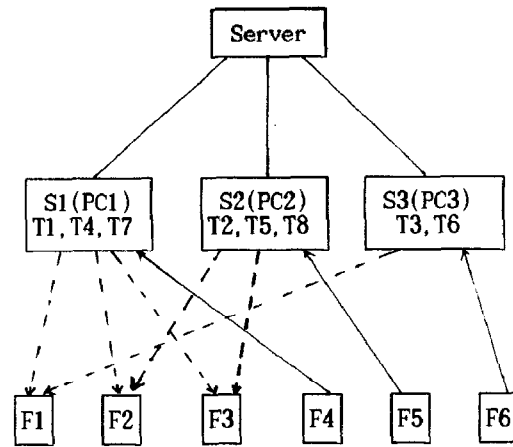


Fig 7. Allocation of the commonly accessed fragments

Now we consider the fragments that are accessed commonly. They can be allocated to a database server or to all the sites requesting them. If the commonly accessed fragments are allocated to a database server, queries are sent to the server and processed there, and then the result of the queries are sent back to the requesting sites. The cost for this scenario depends on the size of results, the frequency of requests, etc. On the other hand, if the commonly accessed fragments are replicated and allocated to all the requesting sites, we should consider the update cost for updating because an update query has effect on all copies of a fragment. Since update cost is determined by the number of replication and the unit cost per update, there is a trade-off between the transmission cost for results and update cost for the allocation of the commonly accessed fragments. For example, Figure 7 shows the environment of the allocation of the commonly accessed fragments in Figure 6. Note that fragment F₁ is requested from both site S₁ by transaction T₄ and site S₃ by transaction T₆ (See Figure 4). Suppose that F₁ be allocated to the server. Then to perform T₄ we first send T₄ to the server, and send the results of T₄ back to S₁. Update transaction T₆ is performed on the server and no side effect is produced. If, however, we suppose that F₁ be allocated to both sites S₁ and S₃, then T₄ is performed at S₁ but update transaction T₆ gives rise to the updating for both copies of F₁ to perform update operations.

4. CONCLUSION

In this paper we investigated several factors that should be considered for the development of effective PC-based distributed database systems. They are data fragmentation, data allocation, LAN, and database server. Then we proposed a methodology for PC-based distributed database design. This methodology, which is based on the mixed partitioning technique using a grid approach, can be done in the following two steps: (1) Grid optimization, (2) Allocation of the mixed fragments. For grid optimization, we developed a cost model and proposed a heuristic procedure for merging grid cells. The results of grid optimization

is called "mixed fragments" and are allocated to the PCs on the corresponding sites by analyzing transactions. The proposed distribution scheme can improve the system performance by allowing data sharing among PCs and by optimizing total transaction processing cost.

Our work can be extended by incorporating performance evaluation methods for merging grid cells. We will also continue to study the architecture and functions that a database server should have.

REFERENCES

[Aper 88] Apers, P. M. G., "Data Allocation in Distributed Database Systems," ACM Trans. on Database Systems, Vol. 13, No. 3, pp.263-304, September 1988.

[Cart 84] Carter, G., Local Area Network, ICL, 1984.

[Ceri 82] Ceri, S., Negri, M., and Pelagatti, G., "Horizontal data Partitioning in Database Design," Proc. ACM SIGMOD International Conference on Management of Data, pp.128-136, 1982.

[Cer 83a] Ceri, S., and Navathe, S. B., "A Methodology to the Distribution Design of Databases," Proc. IEEE COMPCON Conference, San Francisco, CA., February 1983.

[Cer 83b] Ceri, S., Navathe, S. B., and Wiederhold, G., "Distribution Design of Logical Database Schemas," IEEE Trans. on Software Engineering, Vol. SE-9, No. 4, pp.487-504, July 1983.

[Ceri 84] Ceri, S., and Pelagatti, G., Distributed Databases : Principles and Systems, McGraw Hill, 1984.

[Ceri 88] Ceri, S., Perinici, B., and Wiederhold, G., "Optimization problems and Solution Methods in the Design of Data Distribution," Working Paper, Stanford University, 1988.

[Corn 87] Cornell, D. W., and Yu, P. S., "A Vertical Partitioning Algorithm for Relational Databases," Proc. Third International Conference on Data Engineering, pp.30-35, February 1987.

[Corn 88] Cornell, D. W., and Yu, P. S., "Site Assignment for Relations and Join Operations in the Distributed Transaction Processing Environment," Proc. Fourth International Conference on Data Engineering, Los Angeles, February 1988.

[Corn 89] Cornell, D. W., and Yu, P. S., "On Optimal Site Assignment for Relations in the Distributed Data Environment," IEEE Trans. on Software Engineering, Vol. 15, No. 8, pp.1004-1009, August 1989.

[Elma 89] Elmasri, R., and Navathe, S. B., Fundamentals of Database Systems, Benjamin /Cummings Publishing, 1989.

[Hamm 79] Hammer, M., and Niamir, B., "A Heuristic Approach to Attribute Partitioning," Proc. ACM SIGMOD International Conference on Management of Data, May 1979.

[Jord 90] Jordan, L., and Churchill, B., Communication and Networking for the IBM PC and Compatibles, Third Edition, Brady Books, New York, 1990.

[Nava 84] Navathe, S. B., Ceri, S., Wiederhold, G., and Dou, J., "Vertical Partitioning Algorithms for Database Design," ACM Trans. on Database Systems, Vol.9, No.4, pp.690-710, December 1984.

[Nava 89] Navathe, S. B., and Ra, M., "Vertical Partitioning for Database Design: A Graphical Algorithm," Proc. ACM SIGMOD International Conference on Management of Data, pp.440-450, May 1989.

[Nava 90] Navathe, S. B., Ra, M., Varadarajan, R., Karlapalem, K., and Streewastav, K., "A Mixed Partitioning Methodology for Distributed Database Design," UF-CIS TR 90-17, University of Florida, 1990.

[Ozsu 91] Ozsu, M. T., and Valduriez, P., Principles of Distributed Database Systems, Prentice Hall, 1991.

[Patt 90] Pattipatti, K. R., "A File Assignment Problem Model for Extended Local Area network Environments," The 10th International Conference on Distributed Computing Systems, Paris, France, May 1990.

[Ra 90] Ra, M., "Data Fragmentation and Allocation Algorithms for Distributed Database Design," Doctoral Dissertation, University of Florida, 1990.

[Ra 91] Ra, M., "A Graph-based Horizontal Partitioning Algorithm for Distributed Database Design," Journal of the Korea Information Science Society, Vol. 19, No. 1, 1992.

[Rein 89] Reiner, D. S., "PC-based Database Management System," ACM SIGMOD Tutorial, May 1989.

[Yu 85] Yu, C. T., Suen, C., Lam, K., and Siu, M. K., "Adaptive Record Clustering," ACM Trans. on Database Systems, Vol. 10, No. 2, pp.180-204, June 1985.