Physical Design of Network Model Databases Using the Property of Separability

Kyu-Young Whang*
Gio Wiederhold

Stanford University

Daniel Sagalowicz

SRI International

Abstract

A physical design methodology for network model databases is developed using the theory of separability. In particular, a large subset of practically important access structures provided by network model database systems is shown to have the property of separability under the usage specification scheme proposed. The theory of separability was introduced in an earlier work, in the context of relational systems, as a formal basis for partitioning the problem of designing the optimal physical database. The theory proves that, given a certain set of access structures and a usage specification scheme, the problem of optimal assignment of access structures to the entire database can be reduced to the subproblem of optimizing individual record types independently of one another. The approach presented significantly reduces the complexity of the design problem which has the potential of being combinatorially explosive.

1. Introduction

Performance is an important issue in designing databases. As a result, the problem of physical database design has been given much attention in recent years. This problem concerns finding an optimal configuration of physical files and access structures – given the logical access paths that represent the interconnection among objects in the data models, the usage pattern of those paths, the organizational characteristics of stored data, and the various features provided by a particular database management system (DBMS) [HSI 70] [CAR 75] [SCH 75] [SEV 75] [HAM 76] [YAO 77] [BAT 80] [GER 77] [GAM 77]. Throughout this paper, we use the term *access structure* as a generic term for both access methods (e.g., indexes) and storage structures (e.g., various strategies for the placement of records) that a particular DBMS provides. In the physical database design, access structures are specified to support logical objects (such as record types or the entire database) in the database. We use the term *access configuration* of a logical object to mean the aggregate of access structures specified to support that logical object.

In the past, most of the research on this subject concentrated on rather simple cases dealing with a single file; in many cases, such a file represents the storage structure for one logical object (such as a relation in the relational model or a record type in the network model). In a database organization, however, the access configurations for many logical objects have complex interrelationships and access patterns. A simple extension of single-file analyses does not suffice for understanding the interactions among logical objects.

Some efforts have been devoted to the cases of multiple logical objects [GER 77] [BAT 80] [KAT 80]. The approaches employed, however, either fall short of accomplishing automatic design of optimal physical databases or provide only general, not quantitative methods. Cost models were developed in [GER 77] and [BAT 80], but it is difficult to use them for the optimal design of physical databases without an exhaustive search among all possible access configurations of the database. (A method based on heuristic pruning of the search space has been reported in [SCH 79].) As pointed out in [GER 77], a relevant partitioning of the entire design is necessary to make the optimal design of physical databases a practical matter.

The theory of *separability*, which was used in [WHA-a 81] for the physical design of relational databases, can be employed for network model databases as well. The theory proves that, if certain conditions are satisfied, the problem of designing the optimal physical database can be reduced to the subproblem of optimizing individual record types independently of one another. Once the problem has been partitioned, the techniques developed for single-file designs can be applied to solve the subproblems. The conditions to be satisfied, however, are general in nature, and their details must be analyzed for individual systems to be considered.

We shall develop, in this paper, a physical design methodology for network model databases using the property of separability. Since network model database systems provide different variety of access structures and have different characteristics (e.g., they are more procedural in nature) than relational systems do, we need to set up a fairly different framework (especially usage specification) for the development of a design methodology. Therefore, we shall put emphases on developing a usage specification scheme that is suitable for describing the network model database environment and on proving that, under this usage specification scheme, a large subset of practically important access structures that are available in

*Authors' current addresses: Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, Stanford, CA 94305, and Artificial Intelligence Center, SRI International, Menlo Park, CA 94025

network model database systems satisfies the conditions for separability. This design procedure based on the property of separability will then be extended, using heuristics, to include other access structures that are not considered initially. We discuss the issues involved in designing the access configuration of a physical database so as to minimize the number of disk accesses for a set of read and update transactions that act upon it.

We choose the system specification given in the Journal of Development of CODASYL Data Description Language Committee [COD-a 78] and that of CODASYL Cobol Committee [COD-b 78] (CODASYL '78 Database Specification) as our environment. Features provided in this report will be briefly introduced in Section 2. Section 3 introduces key assumptions, while Section 4 describes the principle of separability and the design theory. A design algorithm based on the theory will be introduced in Section 5. Extensions of our approach are mentioned briefly in Section 6.

2. CODASYL '78 Database Specification

In this section, we introduce the features provided by the CODASYL '78 Database Specification. (We use the '78 description to handle a broader spectrum of access structures that may be used in network model database systems. The '71 version can be treated in a similar but easier way.) In this new specification, the concept of storage schema has been introduced to separate many storage-related aspects from the conceptual schema. The storage schema is defined by using the Data Storage Description Language (DSDL) which is separate from the Data Description Language (Schema DDL). Among many new features, the following ones are of interest in the physical database design: (Note that the DSDL in [COD-a 78] was only a proposed draft. In our discussion, however, we keep using this version as a model for network model database systems.)

- The schema DDL now allows multiple *record keys* to be defined for each record type. A record key is called a *record-ordering key* if an order is defined for it by specifying ASCENDING or DESCENDING.

- Indexes can be defined in the storage schema to support the record keys specified in the conceptual schema. Indexes can also be used to represent a SET type, i.e., as pointer arrays. (Throughout this paper, the term *SET* will be used to mean a DBTG set.)

- A serial scan of all the records of a record type is possible by specifying a *record-order key* in the subschema, which in turn should be mapped to a record-ordering key in the conceptual schema. Only one record-order key can be defined in the subschema. Serial order here implies only a logical ordering and does not necessarily mean that the records are actually stored sequentially.

- The placement of a record (location mode in earlier terms) can be done in any one of three different ways. (We ignore secondary options such as DISPLACEMENT or WITH in the DSDL.) A record can be

  - Placed according to a CALC key,

  - Clustered via a SET defined in the conceptual schema, and, optionally placed near the owner,

  - Stored sequentially in ascending or descending order according to the value of a set of data items.

3. Assumptions

In this section, we summarize the key assumptions that will be used throughout the paper.

The database is assumed to reside on disklike devices. Physical storage space for the database is divided into fixed-size units called blocks [WIE 77]. The block is not only the unit of disk allocation, but also the unit of transfer between the main memory and the disk.

We assume that records of all types are stored in one area, and that they are randomly scattered therein. It is assumed that the clustering of records of the member type of a SET affects the relative distances between records of that type, but does not affect the distances between records of other types. To make this assumption valid, we exclude the clustering of member records near their owner record.

We assume that the CALC records are randomly distributed, and that the average number of block accesses required to access one record by CALC key is the same for any record type and for any key, depending only on the overall load factor of the area.

We ignore any disparity in the size of records of the owner type of a SET that results from various SET implementations, so that a SET implementation affects the size of member records only (because of the space needed for additional pointers). Furthermore, if an index is used to represent a SET occurrence, it is assumed that this index is not stored near the owner record (i.e., the NEAR OWNER option for the placement of index entries is excluded from our consideration).

A multimember SET and other options, such as sorted SETs, will not be considered.

4. Design Theory

In this section, we develop the design theory based on the concept of separability. Specifically, we introduce the formal definition of separability, formulate the partial-operation cost, and show that the model system (which will be defined in Section 4.2) consisting of a subset of access structures in CODASYL '78 Database Specification, satisfies the separability under the assumptions we made in Section 3 and the usage specification we

shall develop. A cost model similar to the one developed by Gerritsen [GER 77] is introduced as an example of a separable system. Finally, update costs are discussed briefly.

4.1. Theorem of Separability

Definition 1: The procedure of designing the optimal access configuration of a network model database is *separable* if it can be decomposed into the tasks of designing the optimal configurations of individual record types independently of one another. □

Definition 2: A *partial-operation cost* of a transaction is that part of the transaction-processing cost that represents the accessing of only one record type, as well as of the auxiliary access structures defined for it. □

We consider various SET implementations to be the access structures that belong to their member record type. Accordingly, the access cost of owner records, when they are accessed through the SET, will be included in the partial-operation cost for the member record type as shown in Equation 5.

Definition 3: A *partial operation* is a conceptual division of the transaction whose processing cost is a partial-operation cost. □

Theorem 1: The procedure of designing the optimal access configuration of a network model database is separable if the following conditions are satisfied:

1. The partial-operation cost of a transaction for a record type can be determined regardless of the access configuration specified for and the partial operation used for the other record types.

2. A partial operation for a record type can be chosen regardless of partial operations used for the other record types.

Proof: Condition 2 states that, in selecting a partial operation of a transaction for a record type, we are not constrained by the partial operations chosen for the other record types. Furthermore, since a partial-operation cost of a record type is not affected by the access configurations of and the partial operations used for the other record types, neither the specific access structures assigned to one record type nor the partial operation used for it can affect any design parameters for other record types. It is therefore guaranteed that there will be no interference among the designs of individual record types. □

4.2. Access Structures in the Model System

Our model includes the following access structures:

1. Placement by a CALC key

2. Placement by CLUSTERING VIA SET (but not NEAR OWNER)

3. Indexes

4. Singular SETs

5. Record-order key

6. Various SET implementations

   a. Link with next pointer

   b. Link with next pointer and prior pointer

   c. Link with next pointer, prior pointer, and owner pointer

   d. Link with next pointer and owner pointer

   e. SET implementation by index (pointer array)

Although a singular SET is specified in the conceptual schema, it is an option that can be used to improve the performance. Thus, we view it here as an access structure available for the physical database design. The record-order key defined in the subschema is likewise regarded as an access structure.

The placement strategies

1. SEQUENTIAL

2. CLUSTERED VIA SET NEAR OWNER

are not included here, since, in the following situations, a condition for separability is not satisfied:

Situation 1: In Figure 4-1 we have two record types, R_1 and R_2, that are the owner and the member types, respectively, of SET type S. The symbol $--*$ in the figure represents a SET type and the asterisk refers to the member record type. It is desired, while a transaction is being processed, that SET type S be traversed from R_2 to R_1 for every record in R_2, and that the R_2 records be scanned according to their physical order. The R_2 records are stored sequentially (by the SEQUENTIAL option) according to the values of the data items whose values determine the set membership (linking data items). (Linking data items correspond to the join attributes in relational terms.)

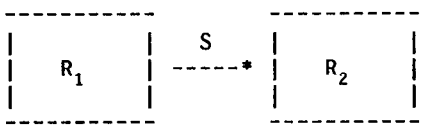

Figure 4-1: Record Types R_1, R_2 and SET Type S between Them

In this situation, the order of accessing the records of R_1 will be random if R_1 records are not stored sequentially (by the SEQUENTIAL option) according to the values of the linking data

items. Randomly accessing R_1 records will result in approximately one block access for every record of R_2. However, if the records of R_1 are stored sequentially, the records of R_1 will be accessed in the order of physical address, resulting in far fewer block accesses. Thus, the partial-operation cost for R_2 (note that the cost of accessing R_1 records as owners through a SET is included in the partial-operation cost for the member record type R_2) is dependent on the access structure of R_1 (i.e., depends on whether or not R_1 is stored sequentially), which violates the condition for separability. □

Situation 2: Figure 4-2 describes four record types, R_0, R_1, R_2, and R_3. Set types S_1, S_2, and S_3 are defined among them. If the placement of R_2 records is declared as CLUSTERED VIA SET S_2 NEAR OWNER, then R_2 records will be clustered around R_1 records. Similarly, if the placement of R_3 records is declared as CLUSTERED VIA SET S_3 NEAR OWNER, then R_3 records will be clustered around R_1 records. Let us assume that the placement strategy of R_1 records is CLUSTERED VIA SET S_1. Then the accessing of the member records (R_1 records) of an occurrence of SET S_1 will be different, depending on whether R_2 or R_3 is clustered via its SET (S_2 or S_3) near the owner R_1, since the intervening records will affect the distances between R_1 records. Thus, the partial-operation cost for R_1 is dependent on the access configurations of R_2 or R_3, which violates the condition for separability. □

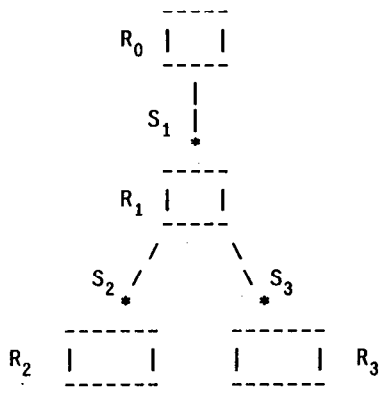

Figure 4-2: Record Types R_0,R_1,R_2,R_3 with SET Types S_1,S_2,S_3

In the model introduced in this section, a significant portion of the access structures provided by the CODASYL '78 Database Specification is included. Those access structures excluded will be incorporated by a heuristic extension.

4.3. Usage Specification

The problem of designing an optimal physical database for network model systems is difficult because of the intrinsic procedural elements in those systems. Thus, once a physical database is designed according to a certain usage specification in a procedural form, there is a possibility that the usage pattern will change as users perceive a new physical structure. This happens because the usage specification in a procedural form does not necessarily represent the optimal translation of the nonprocedural specification. Nor can we get the optimal translation before we have the specific physical database structure. (This is the classic chicken-and-egg problem.) Although the cycle may converge to some local optimum, the true optimum cannot be achieved.

Another difficulty with procedural specifications stems from data dependencies. As an example, let us assume that a record key is defined in the conceptual schema and the subschema and that the programs use it explicitly. This key cannot then be eliminated without changing all the programs that use it. Similarly, once a singular set is defined in the schema and used by programs, it cannot be eliminated without changing these programs. In the system described by the DBTG proposal [COD 71], once a CALC key has been defined and used in application programs, it cannot be redefined without jeopardizing those programs.

One possible approach to averting all these problems would be to employ a nonprocedural usage specification. We would then have to have a hypothetical optimizer to translate the transaction in a nonprocedural form into an optimal sequence of operations. In principle, the design can be accomplished as follows:

- Enumerate all possible access configurations of the physical database
- Using the hypothetical optimizer, evaluate the minimum possible processing cost for each configuration
- Find out the access configuration that yields the minimum cost.

If we design the optimal physical database structure, initially, based on a nonprocedural usage specification, the application programs will adapt themselves towards the true optimum. A good initial design is particularly important when a full data independence is not provided by the system.

We choose here a scheme for the usage specification that is rather nonprocedural and is similar to the approach used in [GER 77]. The usage is divided into 2 classes: one is the usage representing the entry to the database, the other the traversal of SETs, in which all the interactions among the different record types are reflected. For the SET traversal, the directions of the traversal (i.e., owner to member or member to owner) are explicitly specified in the usage. On the other hand, all the processing for the database entry is subject to optimization. Thus, for each operation, a decision has to be made as to which key is to be used (if the operation has a predicate that matches more than one key), whether a scan using the record-order key or the singular set is to be performed, etc., so as to yield the minimum cost. The fixed direction of a SET traversal is necessary to make the design separable, since, otherwise, both directions have to be considered, and the choice of the direction will depend on the access configurations of both record types.

The two classes of usage information are as follows:

- For database entry
  - f_{ENT} *(T, R, PRED)* is the frequency of entry to the record type R in processing the transaction T. PRED represents the predicate, which is in the conjunctive normal form, to be applied to the record type R. A *simple predicate* is an equality predicate on one data item, such as DATAITEM = DATAVALUE. A *candidate key* is defined as the list of all data items, each of which appears in a conjunct of PRED that is a simple predicate. Only candidate keys are considered as potential record keys to be supported in the storage schema.
- For SET traversal
  - f_{OM} *(T, R, S, PRED)* is the frequency of traversal of SET type S, in processing transaction T, from the owner to the member (record type R). PRED is the predicate to be applied to the owner record type.
  - f_{MO} *(T, R, S, PRED)* is the frequency of traversal of SET type S, in processing transaction T, from the member (record type R) to the owner. PRED is the predicate to be applied to the member record type. These parameters are illustrated in Figure 4-3.

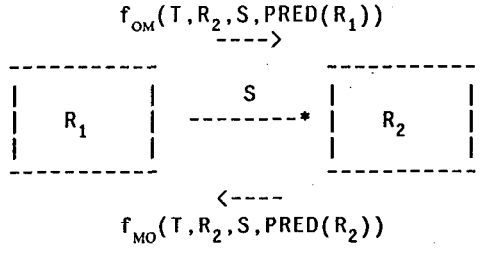

Figure 4-3: Usage Parameters for SET Traversal

4.4. Formulation of Partial-Operation Costs

To formulate the partial-operation cost, we develop the following notation.

Elementary-Operation Costs

C_{ENT}(R, PRED, candidate-key)
: The cost of scanning the records of type R using the candidate-key with predicate PRED.

C_{SCAN}(R, singular-set)
: The cost of scanning the records of type R using a singular set.

C_{SCAN}(R, record-order-key)
: The cost of scanning the records of type R using the record-order-key. The predicate is not resolved before the record is fetched.

C_{SCAN}(R, area-scan)
: The cost of scanning the records of type R by scanning the whole area.

C_{OM}(R, S)
: The cost of traversing one SET occurrence of type S from the owner record to its member records (of type R). The cost of accessing the owner record is excluded since the owner record must have been accessed through other access structures that belong to the owner record type.

C_{MO}(R, S)
: The cost of accessing member records and the owner record when traversing one SET occurrence of type S from a member record (of type R) to its owner. The starting member record is assumed to have been accessed already.

Usage-Transformation Functions

In Section 4.3, the usage associated with SETs was specified as the frequencies of traversals of SET types. This must be translated into the frequencies of traversals of SET occurrences. We need the following definition and notation: (The usage transformation scheme that will be described here is suitable for the queries of two record types. The usage specification for the queries of more-than-two record types is currently being developed. It mainly has to deal with predicate branches in the query graph.)

Definition 4: The *linkage factor* $J_{R,S}$ of a record type R with respect to a SET type S is the ratio of the number of records of type R that are linked in any occurrence of S to the total number of records of type R. (This is similar in concept to the join selectivity in relational systems [WHA-a 81].) □

n_R : Number of records of record type R (cardinality).

$g_{R,S}$: Number of member records (of type R) in a SET occurrence of SET type S (grouping factor).

owner(R,S) : The owner record type of SET type S whose member record type is R.

SEL(PRED,R) : Selectivity of predicate PRED when applied to the records of type R.

We now define three usage-transformation functions in accordance with three SET-related usage parameters.

$$F_{OM}(f_{OM}, T, R, S, PRED) = f_{OM}(T, R, S, PRED) \times n_{owner(R,S)} \times SEL(PRED, owner(R,S)) \times J_{owner(R,S),S} \quad (1)$$

$$F_{MO}(f_{MO}, T, R, S, PRED) = f_{MO}(T, R, S, PRED) \times b((n_R \times J_{R,S})/g_{R,S}, g_{R,S}, n_R \times SEL(PRED, R)) \quad (2)$$

Here the function b(m,g,k) computes the number of record groups selected, where k is the number of records selected, g the number of records in a group, and m the total number of record groups considered. In the form in Equation 2, the b function gives the number of set occurrences that have at least one member record (of type R) satisfying predicate PRED. An exact form of this function and various approximation formulas are summarized in [WHA-b 81]. It is approximately linear in k when $k \ll n$ ($n = m \times g$), and approaches m as k becomes larger. A familiar approximation suggested by Cardenas [CAR 75] is $b(m,g,k) = m\,[1 - (1 - 1/g)^k]$.

Partial-Operation Cost

Given an access configuration of the physical database, the partial-operation cost of transaction T for record type R will be

$$POC(T, R) = Cost_{DB\text{-}ENTER}(T, R) + Cost_{SET\text{-}TRAVERSE}(T, R), \quad (3)$$

where

$$Cost_{DB\text{-}ENTER}(T, R) = \sum_{PRED} \min \{ \quad (4)$$

$$\min_{candidate\text{-}key}\{f_{ENT}(T,R,PRED) \times C_{ENT}(R,PRED,candidate\text{-}key)\},$$
$$f_{ENT}(T,R,PRED) \times C_{SCAN}(R,singular\text{-}set)^{\dagger},$$
$$f_{ENT}(T,R,PRED) \times C_{SCAN}(R,record\text{-}order\text{-}key)^{\dagger},$$
$$f_{ENT}(T,R,PRED) \times C_{SCAN}(R,area\text{-}scan)\},$$

and

$$Cost_{SET\text{-}TRAVERSE}(T, R) = \sum_{S \in \{SET\ types\ whose\ member\ is\ R\}} \sum_{PRED} \{ \quad (5)$$

$$F_{OM}(f_{OM}, T, R, S, PRED) \times C_{OM}(R, S) +$$
$$F_{MO}(f_{MO}, T, R, S, PRED) \times C_{MO}(R, S)\}.$$

Entries in Equation 4 marked with the symbol † are considered only when the corresponding access structures (singular set or record-order key) are available in the given access configuration.

4.5. Separability for the Model System

To verify that the design of the physical database for our model is indeed separable, we have to show that the partial-operation cost POC(T,R) for record type R is independent of the access structures chosen for the other record types. For this purpose, we shall consider each individual component of the partial-operation cost. First, as shown in Equations 1 and 2, the usage-transformation functions are independent of access structures. They depend solely on the characteristics of the data such as the cardinality of a record type, linkage factors, grouping factors, or the selectivity of a predicate for a record type, etc. (These are already known at design time.)

Elementary-operation costs C_{ENT} and C_{SCAN} for a record type, say R, are not affected by the access structures of record types other than R. We reason as follows:

- Entering the database through record type R accesses only records of type R.
- The records to be actually accessed and the order of accessing them are determined by the characteristics of the access structures of R itself.
- In accordance with our assumption in Section 3, clustering of member records (but not near owner), access structures such as indexes, or various SET implementations in any record type other than R, do not affect the relative distances of records of type R.
- The accessing cost when using a CALC key is not affected by any access structures, since, on the basis of our assumption in Section 3, this will depend solely on the load factor of the area.

$C_{OM}(R, S)$, the cost of accessing the member records (of type R) of one SET occurrence when it is traversed from the owner to members, is dependent only on the access structures (e.g. SET implementation or clustering) of R itself, because of reasons similar to those above.

$C_{MO}(R, S)$, which is the cost of accessing member records and the owner record of one SET occurrence when it is traversed from a member to the owner, consists of two components: the cost of accessing the member records and the cost of accessing the owner record. The former can be explained as in the previous case (C_{OM}). The latter depends on whether the records of type R (which is the member) are clustered on the linking data items. If so, the same SET occurrence and accordingly the same owner record will be accessed consecutively. The owner record may well stay in the buffer and cause one block access for one set occurrence. However, if the member records are not clustered, a SET occurrence can be traversed repeatedly in a random order (i.e., not consecutively) and will cause one block access to access the same owner record for each traversal of the SET occurrence. Thus, the cost of accessing the owner records through a SET is dependent on the access structures for its member record type. This is why we included that cost in the partial operation cost for the member record type. Let us note that this cost does not depend on the access structures of the owner type.

Since all the components of the partial processing cost for record type R are independent of the access structures of other record types, so is the partial processing cost, thus satisfying Condition 1 for separability.

Condition 2 in Theorem 1 is satisfied, since we are not restricted at all in our choice of elementary operations (for database entering or SET traversing) for a record type by choices made for other record types. (This condition may be a significant restriction upon relational systems, especially in the selection of join algorithms.)

Hence, we conclude that entire design procedure for our model system is separable.

4.6. Example Cost Model

As an example, let us investigate the cost model developed by Gerritsen [GER 77]. Based on a similar system [GER 76] described in the DBTG proposal [COD 71], it is presented here in a slightly modified form incorporating the following assumptions to be consistent with the assumptions we have used:

- Member records of a SET occurrence cannot be clustered near their owner.
- All the records of any type are stored in one area.
- The difference between sequential and random block accesses is ignored, so that the cost measure is simply the number of block accesses.
- Predicates are normally assumed to qualify more than one record, so that all the records of a type have to be accessed when they are scanned. (If it is known that only one record satisfies the predicate, only about half the records, on the average, will have to be accessed, which is the only case considered in [GER 77].)

The following notation will be used in the cost model:

$x_{R,S}$	1 if the placement strategy of record type R is CLUSTERED VIA SET S, and 0 otherwise.
Z_R	Size in bytes of a record of type R.
B	Size in bytes of a block.
LF	Load factor of the area in which the database is stored. Here it is assumed to be constant throughout the design procedure.
$g_{R,S}$	Number of records of type R (which is the member) in a SET occurrence of type S.
Q_S	1 if SET type S has the owner pointer, and 0 otherwise.
n_R	Number of records of type R (cardinality).
P	Number of blocks in the area.
f	An overflow function indicating the average number of block accesses, in excess of 1, required to retrieve a record by a CALC key.

We define MAC(R, S) (member-accessing Cost) as the expected cost of completing the physical accesses required to visit all the members (of type R) of an occurrence of SET type S. Then we have

$$MAC(R, S) = x_{R,S} \times \lceil (Z_R \times g_{R,S}) / (B \times (1 - 0.5\ LF)) \rceil + (1 - x_{R,S}) \times g_{R,S}.$$

The first term calculates the average number of blocks touched when the placement strategy for record type R is CLUSTERED VIA SET S. The factor 0.5 in the denominator is for adjusting the load factor. This factor is obtained based on the assumption that the load factor is 0 when the first SET occurrence is loaded, whereas it is LF when the last SET occurrence is loaded. The second term represents the cost when the placement strategy of the record type R is not CLUSTERED VIA SET S, in which case the records in a SET occurrence are accessed randomly.

Using MAC(R,S), we can obtain the elementary-operation costs as follows:

$$C_{ENT}(R, \text{Calc-key}) = 1 + f(LF) \quad (6)$$

$$C_{SCAN}(R, \text{singular-set}) = n_R \quad (7)$$

$$C_{OM}(R, S) = MAC(R, S) \quad (8)$$

$$C_{MO}(R, S) = (1 - Q_S) \times 0.5 \times MAC(R, S) + 1 \quad (9)$$

In Equation 9, it was assumed that accessing the owner record causes one block access for each traversal of a SET occurrence regardless of whether a SET occurrence is traversed consecutively or randomly.

We note that all the elementary costs in this model for record type R are independent of access configurations for the other record types; consequently, the design is separable.

4.7. Update Cost

Although a detailed usage specification for update transactions will not be developed here, the following points are worth noting.

An update operation can be viewed as a series of operations that locate the record to be updated as well as those that are accessed on the way of locating it. Thus, usage specifications similar to the ones used in previous sections can be employed for the updates.

As mentioned previously, we included the cost of accessing the owner record through a SET in the partial-operation cost of the member record type. By the same token, the cost of updating the pointers (used for a specific SET implementation) of the records of the owner record type of a SET must be included in the partial-operation cost of the member record type. This is because the cost is a function of the specific SET implementation, and the SET implementation is regarded as an access structure of the member

record type.

It is not difficult to conclude that, if we segregate the costs into partial-operation costs for each record type as defined in Section 4, update costs for a record type other than that of SET pointers will not be affected by the access configurations for the other record types,

5. Design Algorithm

In this section, an algorithm for the design of optimal access configurations will be presented. Based on the result of Theorem 1, the algorithm is as follows:

Inputs:

- Usage information: f_{ENT}, f_{OM}, f_{MO} as defined in Section 4.3, for each transaction, record type, SET type, and predicate, together with their respective frequencies. Usage specification of update transactions with their frequencies.

- Data characteristics: for each record type—its cardinality, the size of a record, selectivity of the domain of each data item, the grouping and linkage factors of a record type with respect to the SET types connected to it. The conceptual schema specifying SET types defined among record types, SET selection strategy, etc.

Algorithm:

1. Using the given usage information and data characteristics, evaluate the usage-transformation functions (F_{OM}, F_{MO}) for every transaction, record type, SET type, and predicate.

2. Pick one record type and determine the optimal access configuration as follows:

   a. Pick one possible access configuration of the record type.

   b. Given that access configuration, identify the best processing method for each elementary operation (corresponding to an elementary-operation cost) and calculate its cost.

   c. Calculate the partial-operation cost of each transaction. This is done by summing up all the elementary costs identified in Step b—multiplied by their respective frequencies—and all the costs incurred by the update transactions acting upon this record type.

   d. Repeat Steps b and c for all possible access configurations for the record type under consideration. Then determine the one that gives the minimal cost as the optimal configuration for that record type.

3. Step 2 is repeated for every record type in the database. The aggregate result for all record types constitutess he global optimum.

The designer could perform Step 2 in the Design Algorithm by the designer by trying each access configuration with a trial-and-error method. This is similar to the approach used in [GAM 77], except that now we are considering only one record type at a time.

Since there are many different access structures to be chosen, however, even for one record type, enumeration of all the possible access configurations could be an extensive procedure. An alternative approach is to partition the single record type design into several substeps, using heuristics if necessary, with well-defined interfaces. As exemplified in [WHA-a 81] for a relational system, the design procedure could be partitioned into the following two substeps:

- Determination of the placement strategy (this corresponds to *clustering* in relational systems) such as CALC and CLUSTERED VIA set-name, where set-name stands for any SET type whose member is the record type under consideration.

- Selection of auxiliary access structures such as indexes, singular sets, and the record-order key.

This approach should be explored in more detail in the future.

6. Extensions for the Other Access Structures

An extension of the access structures not included in the basic design methodology, such as SEQUENTIAL and CLUSTERED VIA SET NEAR OWNER, can be accomplished by using heuristic methods.

After the basic design is obtained by using the Design Algorithm in Section 5, the SEQUENTIAL option can be considered for each record type (only one at a time is endowed with this property). The total costs with or without this option are compared and the differences calculated for each record type. (Since the SEQUENTIAL structure may affect the partial-operation cost for other record types, the total processing cost has to be considered for comparison.) Record types must be ranked in importance according to the cost differences. The record type that yields the greatest benefit is assigned the top rank. The placement strategy is then actually changed to SEQUENTIAL—if the total cost is reduced—starting with the top-ranked record type.

Another approach for the SEQUENTIAL option is to include it in the basic design methodology, pretending that the design is separable and sacrificing slightly the rigorousness of the property of separability. The major prospects for this option will be record types that require frequent scanning of all their records. But this type of operation does not impair separability, so that we can keep the error minimal while pretending that the design is separable.

The CLUSTERED VIA SET NEAR OWNER option can be

considered next. For every record type whose placement strategy is CLUSTERED VIA SET, the latter is changed temporarily to CLUSTERED VIA SET NEAR OWNER (only one record type at a time is endowed with this property) and difference in total cost is calculated. As in the case of SEQUENTIAL option, the importance of the record types is ranked. The placement strategy is then actually changed to CLUSTERED VIA SET NEAR OWNER—if there is a cost benefit—starting from the top according to the rank. Constraints can be used here, if desired, that not more than one member record type can be clustered near the same owner record type. This approach is similar to the one in [KAT 80] but uses a more quantitative approach to establish the rank.

More research needs to be done on usage specifications for update transactions and on the design algorithm for a single record type. Now that the whole design has been partitioned to the designs of individual record types, any conventional method devised for a single logical object can be applied here.

7. Conclusion

A physical design methodology for network model databases has been introduced. Our main objective in this method is to establish a formal design methodology, based on the idea of separability, in which a large subset of practically important access structures are included. Then, using heuristic methods, we proceed to extend this basic design to include other access structures that have not been incorporated initially. The CODASYL '78 Database Specification has been used as our environment for our discussion.

We have introduced a usage specification scheme that is suitable for describing the network model database environment. It has been proved that, under this scheme, the selected set of access structures indeed satisfies the conditions for separability.

It has been emphasized that the initial design is especially important in the systems that do not provide a full data independence. Our approach provides a tool to achieve the optimal initial design using a largely nonprocedural usage specification.

In summary, the key contribution intended in this paper is to provide a formal methodology for the physical design of network model databases.

Acknowledgment

This work was supported by the Defence Advanced Research Projects Agency, under the KBMS project, Contract Number N39-82-C-0250. The authors wish to thank the referees for their useful comments which made the final version more comprehensive.

References

[BAT 80] Batory, D. S. and Gotlieb, C. C., "A Unifying Model of Physical Databases," Tech. report CSRG-109, Computer Systems Research Group, University of Toronto, April 1980.

[CAR 75] Cardenas, A. F., "Analysis and Performance of Inverted Database Structures," *Comm. ACM*, Vol. 18, No. 5, May 1975, pp. 253-263.

[COD 71] CODASYL, *Data Base Task Group Report*, ACM, New York, 1971.

[COD-a 78] CODASYL Data Description Language Committee, *Journal of Development*, EDP Standards Committee, Secretariat of Canadian Government, Canada, 1978.

[COD-b 78] CODASYL Cobol Committee, *Journal of Development*, EDP Standard Committee, Secretariat of Canadian Government, Canada, 1978.

[GAM 77] Gambino, T. J. and Gerritsen, R., "A Database Design Decision Support System," *Proc. Intl. Conf. on Very Large Databases*, Tokyo, Japan, IEEE, October 1977, pp. 534-544.

[GER 76] Gerritsen, R. et al., "WAND User's Guide," Decision Sciences Working Paper 76-01-03, Wharton School, Univ. of Pennsylvania, 1976.

[GER 77] Gerritsen, R. et al., "Cost Effective Databse Design: An Integrated Model," Decision Sciences Working Paper 77-12-03, Wharton School, Univ. of Pennsylvania, 1977.

[HAM 76] Hammer, M. and Chan, A., "Index Selection in a Self-Adaptive Database Management System," *Proc. Intl. Conf. on Management of Data*, Washington, D.C., ACM SIGMOD, June 1976, pp. 1-8.

[HSI 70] Hsiao, D. and Harary, F., "A Formal System for Information Retrieval from Files," *Comm. ACM*, Vol. 13, No. 4, February 1970, pp. 67-73, Also see Comm. ACM 13, 4, April 1070, p.266.

[KAT 80] Katz, R. H. and Wong, E., "An Access Path Model for Physical Database Design," *Proc. Intl. Conf. on Management of Data*, Santa Monica, Calif., ACM SIGMOD, May 1980, pp. 22-29.

[SCH 75] Schkolnick, M., "The Optimal Selection of Secondary Indices for Files," *Information Systems*, Vol. 1, March 1975, pp. 141-146.

[SCH 79] Schkolnick, M. and Tiberio, P., "Considerations in Developing a Design Tool for a Relational DBMS," *Compsac*, nov 1979, pp. 228-235.

[SEV 75] Severance, D. G., "A Parametric Model of Alternative File Structures," *Information Systems*, Vol. 1, No. 2, 1975, pp. 51-55.

[WHA-a 81] Whang, K., Wiederhold, G., and Sagalowicz, D., "Separability: An Approach to Physical Database Design," *Proc. Intl. Conf. on Very Large Databases*, Cannes, France, IEEE, September 1981, pp. 320-332.

[WHA-b 81] Whang, K., Wiederhold, G., Sagalowicz, D., "Estimating Block Accesses in Database Organizations – A Closed Noniterative Formula," , submitted for publication, 1981.

[WIE 77] Wiederhold, G., *Database Design*, McGraw-Hill Book Company, New York, 1977.

[YAO 77] Yao, S. B., "An Attribute Based Model for Database Access Cost Analysis," *ACM Trans. Database Systems*, Vol. 2, No. 1, March 1977, pp. 45-67.