THE USE OF A DATABASE MACHINE FOR SUPPORTING RELATIONAL DATABASES*

Jayanta Banerjee and David K. Hsiao

Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

Abstract

One of the goals in the design of database machines of the future is their generality. In addition to being capable of carrying out the common database management functions with high reliability and performance, some of these machines are intended to support more than one data model. A specific database machine, known as the DBC, is intended to support several existing data models. Although the DBC supports many data models, we single out the relational data model for this discussion. In particular, we have tried to concentrate mainly on the subject of database representation and query translation of System R-like database management systems. Some estimates of the storage requirements and performance gains are given in this paper. However, due to limited space, the detailed analysis is shown elsewhere in [22].

Introduction

Advances in technology and database research have prompted considerable attention to the design and implementation of database machines [1,2,3,4]. With the design of a number of database machines either completed or underway [5,6,7,8], there are reasons to believe that the prototype construction of database machines is indeed viable. These machines will perform the basic database management functions with improved reliability and performance as compared to those obtained with software means.

One of the most difficult design decisions that confronts the database machine designer is the type of data structure which should be built into the machines. On the one hand, the designer would like to build into the machine a very elaborate and complex data structure so that it is sophisticated enough to emulate the high level data models such as the relational and CODASYL models, making the need of software support for such models superfluous. On the other hand, the designer would like to build into the machine a very simple and elegant data structure so that its straightforward implementation can lead to a machine with high performance and reliability. Such dichotomy is the main focus of the paper.

In this paper, we shall consider a specific database machine known as the DBC which is capable of supporting multiple data models [5,9,10,11,12]. However, the built-in data structure of the database machine is rather simple and straightforward. We would like to show how the data models, say, the relational data model [13,14], are supported on this machine. We would also mention its performance and its storage requirements for relational databases. It is estimated that the DBC storage requirements may not result in any saving over the conventional computer system. However, the DBC performance may be considerably better than what is achievable on a conventional computer system. In the absence of a large-scale commercial relational database management system, System R [15] has been used for the study. The relational language used, therefore, is SEQUEL 2 [16], which we shall refer to simply as SEQUEL. The study of the database machine in supporting other models such as hierarchical and CODASYL has been documented elsewhere [17,18]. Due to the limited length of this paper, we shall not present these findings here.

The Operating Environment - Front-End Computer and DBC

As a special-purpose computer, the DBC is intended to be used as a back-end machine to a front-end conventional computer. The front-end computer supports all application programs, the operating system and a specialized package called the RDBI (Relational Database Interface). The basic organization of the front-end and back-end computer is depicted in Figure 1.

A user who does not want to make use of the database may simply interact with the operating system of the front-end computer to execute his program. A database user, on the other hand, calls upon the services of the RDBI in order to access the relational database which is stored in the DBC. The user programs that access the database may either be written in the stand-alone version of SEQUEL or in a host programming language which embeds SEQUEL as a data sublanguage. In either case, the SEQUEL statements are identified by the operating system (perhaps, with the aid of a precompiler) and transmitted to the RDBI. The RDBI will then execute the statement by sending appropriate commands to the DBC, collecting the DBC responses in its buffer, and sending the final results back to the operating system.

*The work reported herein was conducted at The Ohio State University and supported by contract N00014-75-C-0573 from the Office of Naval Research.

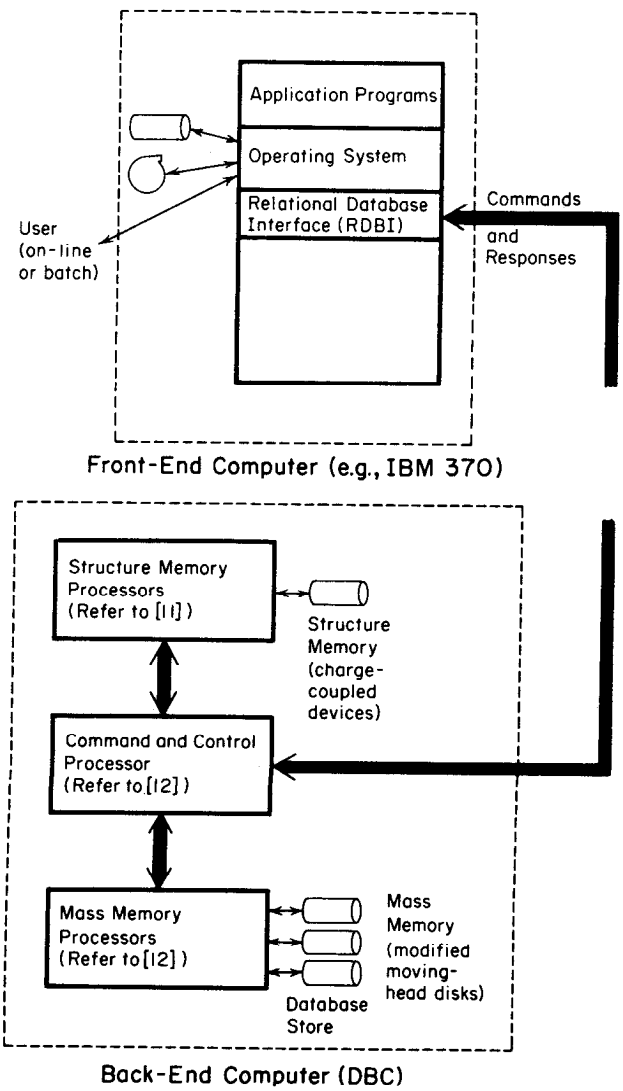

Figure 1. The Basic Relationship of the Front-end and Back-end Computers

The DBC stores the database in an on-line mass memory, which is made of modified moving-head disks. The tracks of a disk cylinder can be accessed and content-addressed simultaneously within a single disk revolution. Access is limited, however, to only one cylinder per drive at a time. Since the number of content-addressable processors is as few as the number of tracks in a given cylinder, the cost reduced in this organization as compared to cost incurred in database machines that associatively address all the cells (e.g., cylinders) may become the single most important factor that makes the DBC possible for very large database stores of the order of 10^{10} bytes. As content-addressable processors, they function simultaneously to search an entire cylinder for records that satisfy a given set of predicates.

Since the on-line mass memory is not a monolithic associative memory, it is important to restrict any database search to as few cylinders as possible. A directory of the database including other structural information of the database (such as security-related information) is, therefore, maintained in a content-addressable memory called the structure memory. The size of this memory is about 1% of the mass memory and is about 20 times faster. Charge-coupled devices (CCDs) are a very cost-effective choice for constructing this memory, as observed in [9,11].

Another major component of the DBC is a controller called the database command and control processor. When a command from the RDBI is sent to the DBC, this controller will decode it, enforce access control by consulting the structure memory, determine the cylinders to be searched with the aid of the structure memory, issue appropriate orders to the mass memory, post-process retrieved data, and transfer the data to/from the RDBI. Although the DBC is well documented elsewhere, this brief outline will be sufficient for our subsequent discussion.

The DBC Data Model

A database in the DBC is a collection of records. A record, in turn, is made up of an ordered set of data items called attribute-value pairs. An attribute-value pair is a member of the product set AT x VA, where AT is a set of "attributes" and VA is a set of "values". Within a record, the attribute part of every attribute-value pair is distinct. The attribute-value pairs that characterize a record (or a group of records) by distinguishing the record (or the group) from all others are called keywords.

A relational operator is an element of the set $\{=,\neq,<,\leq,\geq,>\}$. A triple of the form <attribute, relational operator, value> is called a keyword predicate. A keyword <A,V> is said to satisfy a keyword predicate <Ap,Op,Vp> if and only if A=Ap and V Op Vp, i.e., V and Vp are related by the operator Op. A query is a Boolean expression of keyword predicates in disjunctive normal form. Thus, a query is a disjunction of query conjuncts, which are conjunctions of keyword predicates. A record satisfies a query if it satisfies at least one query conjunct in the query. The set of all records that satisfy a query is called the response set of the query.

As an example of the types of queries that may be recognized by the DBC, consider the following:

([DEPT='TOY']&[SALARY<10000]) v
([DEPT='BOOK']&[SALARY>50000]).

If the above query refers to employee records of a department store, then it will be satisfied by records of the employees working either in the toy department and earning less than 10,000, or working in the book department and making more than 50,000.

Queries are used not only to retrieve a set of records among all the records in a database but also to specify protection requirements and clustering conditions.

DBC Commands

While the DBC is provided with a repertoire

of access and preparatory commands, we shall restrict ourselves here only to a simplified description of the retrieve command (and some of its various forms). This is because in this paper we intend only to illustrate how relational query facilities (or retrieval facilities) are handled in a database managed by the DBC. A description of other relational facilities (such as data control and data manipulation facilities) and their implementation on the DBC will be found in [19]. A detailed description of the DBC commands may be found in [12,20].

A retrieve command has the following two simplified forms, where the square brackets are used as metasymbols to indicate zero or one occurrence of the expression inside them:

Form 1

```
RETRIEVE:[set-function([attribute-1]) [ONLY]]
         ([(UNIQUE)] attribute-list) (query)
         [SORT BY attribute-2]
```

The command requires that the database be first searched to find all records that satisfy the given query. Of the response set, the values will be retained if their attributes appear in the attribute-list (which assumes a '*' if all values, i.e., entire records, are desired). In case the UNIQUE option is specified, then all partial records are discarded. These retrieved records are ordered by attribute-2. The DBC can perform by hardware a number of set functions such as AVG (which computes the average value of the elements in a set), MAX, MIN and SUM (which compute the maximum, minimum and sum, respectively, of the elements in a set). In the retrieve command, attribute-1 refers to the attribute-value pair of each retrieved record, whose attribute part is the same as attribute-1. The set function is performed on all these attribute-value pairs. In case the set function is COUNT, then attribute-1 may be null, in which case the number of retrieved records is counted. The ONLY option is used if the response set of the command is to consist of the set function alone instead of records. Attribute-1 and attribute-2 are both required to appear in attribute-list.

Form 2

```
RETRIEVE:(attribute-list-1) (query-1)
         CONNECT ON (attribute-1,attribute-2)
         (attribute-list-2) (query-2)
```

This command specifies that the set A of records that satisfy query-1 and the set B of records that satisfy query-2 be retrieved. The attribute-value pairs corresponding to attributes in attribute-list-1 are extracted from records of set A to form a set A1 of partial records. Similarly, the attribute-value pairs of attribute-list-2 are extracted from records of set B to form the set B1. An equality join is now made of the two sets of records A1 and B1 to create the final response set. The connecting attributes of the join operation are attribute-1 of set A1 and attribute-2 of set B1. Any record of the response set has three parts: attribute-value pairs corresponding to attribute-list-1 (except attribute-1), attribute-value pair corresponding to attribute-1 and attribute-value pairs corresponding to attribute-list-2 (except attribute-2). Note that it is necessary that attribute-1 be one of the attributes in attribute-list-1 and attribute-2 be one of the attributes in attribute-list-2.

The Relational Data Model

We shall provide here a very brief look at the relational model and the data sublanguage SEQUEL. Conceptually, a relation is a table in which each column corresponds to a distinct attribute and each row corresponds to a distinct entity or tuple. Each tuple is distinct in the sense that no two tuples in a relation have identical values for all attributes. A relation or table in a relational database exists in a normalized form, which means that every column of the table represents a simple attribute and is not itself another relation. Other improvements on the normal form are described in [21].

A comprehensive relational database management system which includes provisions such as simple but flexible user views, data definition, data manipulation and query capabilities, as well as convenient access support, system recovery and integrity enforcement can be found in System R [15]. System R provides user interface through a data sublanguage called SEQUEL [16]. Although the complete collection of System R facilities is available through SEQUEL, we shall concentrate in this paper mostly on the query capabilities, which constitute the most basic operations of the SEQUEL language. A discussion on how the other facilities of SEQUEL are handled in the DBC will be found in [19].

A sample database, extracted from [16], is depicted in Figure 2. It consists of four normalized relations. The EMP relation describes a set of employees, giving the employee number, name, department number, job title, manager's employee number, salary and commission for each employee. The DEPT relation gives the department number, name and location of each department. The USAGE relation describes the parts which are used by the various departments. The SUPPLY relation describes the supplier companies from which the various parts may be obtained. We shall make extensive reference to this sample database in all our later examples.

Relation	Attributes
EMP	EMPNO,NAME,DNO,JOB,MGR,SAL,COMM
DEPT	DNO,DNAME,LOC
USAGE	DNO,PART
SUPPLY	SUPPLIER,PART

Figure 2. A Sample Database

We conclude this section with a short example on the use of the query facilities of SEQUEL. For example, to find the names of employees in Dept. 100, one may write

```
SELECT NAME
FROM   EMP
WHERE  DNO=100
```

The SELECT clause lists the attributes to be returned. If the entire tuple is desired, then one may write SELECT *. The WHERE clause may contain any collection of predicates which compare values of attributes of a tuple to constant values (e.g., DNO=100) or compare values of two attributes of a tuple with each other (e.g., SAL<COMM). The predicates may be connected by AND and OR, and parentheses may be used to establish precedence. In our later example queries, extracted from [16], we shall demonstrate the other varieties of query

facilities in SEQUEL and how these queries are supported by the DBC.

Representing A Relational Database

Each tuple of a relation is stored in the DBC as a single DBC record. Not surprisingly, this record format closely resembles the logical structure of the tuple. While a tuple is normally seen as a sequence of values, where the position of each value identifies the underlying column, it is not sufficient in the DBC to store the values alone for each record. To allow for overlap among the individual domains (of the columns) and because the DBC ignores the absolute positions of the keywords in a record, it is necessary to store the column names as well, within the keywords. Therefore, whenever a tuple is to be stored in the database, the RDBI (relational database interface) creates a DBC record which consists of only keywords. One keyword is created, as shown below, for every column of the relation

<column-name, value>

where the attribute part of the keyword is the name of the column (in a coded form).

In order that the DBC may recognize a tuple of one relation from that of another, an extra keyword, as shown below, is added to each DBC record:

<RELATION, relation-name>

where the value of the keyword is the name of the relation to which the tuple belongs.

Thus, any tuple of the DEPT relation of Figure 2 is represented in the DBC by means of a reocrd with the following attribute-value pairs:

<RELATION, DEPT>
<DNO, department-number>
<DNAME, department-name>
<LOC, department-location>

If any one of the columns does not have a corresponding value in some tuple, then it is not necessary to create (or store) an attribute-value pair for that column. Thus, every DBC record representing a DEPT tuple will have an attribute-value pair for DNO (if this column always takes a non-null value), but it may not have such a pair for LOC (if the department is newly planned and is yet to come into existence).

To improve database performance, the DBC records are primarily clustered according to the keyword with attribute RELATION. That is, all those DBC records that correspond to the tuples of a relation are clustered together. Secondary clusters are formed based on database definition, such as clustering links and clustering images [15].

Translation of SEQUEL Queries

Once the database is created on the DBC by appropriate representation of the relational database, all the normal data management functions may then be carried out by the DBC. Every SEQUEL query received by the RDBI is translated into a sequence of DBC commands, some of which may depend on the results of previous commands within the sequence. In each of the following examples, the statement of a problem is first made, then a SEQUEL statement is written to solve the problem and finally this SEQUEL statement is translated into a sequence of one or more DBC commands. The database referenced is the one shown earlier in Figure 2.

Example 1: The following SEQUEL statement and DBC command will find the names of employees in Dept. 50.

```
SEQUEL:
  SELECT NAME
  FROM   EMP
  WHERE  DNO=50
DBC Command:
  RETRIEVE:  (NAME) ((RELATION='EMP')&(DNO=50))
```

Example 2: To list the names of employees in departments 25, 47 and 53, the following statement may be used.

```
SEQUEL:
  SELECT NAME
  FROM   EMP
  WHERE  DNO IN (25,47,53)
DBC Command:
  RETRIEVE:  (NAME) (((RELATION='EMP')&(DNO=25))
                    v((RELATION='EMP')&(DNO=47))
                    v((RELATION='EMP')&(DNO=53)))
```

Example 3: Consider listing the names of employees who work for departments in Evanston. This type of transaction requires access to two different relations and is, therefore, expressed in SEQUEL by means of a nested SELECT statement. The inner part of the nesting returns the collection of DNO values of the departments located in Evanston. The outer part then proceeds as though it were given a set of constants in lieu of the inner SELECT clause.

```
SEQUEL:
  SELECT NAME
  FROM   EMP
  WHERE  DNO IN
         SELECT DNO
         FROM   DEPT
         WHERE  LOC='EVANSTON'
DBC Commands:
  a.  RETRIEVE: (DNO) ((RELATION='DEPT')&(LOC=
EVANSTON')). For each department number 'di'
retrieved by (a), the RDBI issues the DBC com-
mand:
  b.  RETRIEVE:(NAME) (RELATION='EMP')&(DNO=
'di'))
```

Example 4: An important class of queries is exemplified in the determination of average salary of clerks. The built-in SEQUEL function AVG can be used to accomplish this result. Other built-in functions in the SEQUEL language are SUM, COUNT, MAX and MIN.

```
SEQUEL:
  SELECT AVG(SAL)
  FROM   EMP
  WHERE  JOB='CLERK'
DBC Command:
  RETRIEVE: AVG(SAL) ONLY
            (*) ((RELATION='EMP')&(JOB='CLERK'))
```

Notice that the (*) in the DBC command indicates that entire records must be retrieved before the function AVG is performed. Of course, the same effect could have been achieved by replacing the (*) with (SAL), thereby avoiding the cost of storing entire records in the DBC. The clause ONLY indicates that only the value of the function need be returned to the RDBI.

Example 5: The following statement determines the count of all the different jobs held by employees

in Dept. 50.

```
SEQUEL:
  SELECT COUNT(UNIQUE JOB)
  FROM   EMP
  WHERE  DNO=50
DBC Command:
  RETRIEVE: COUNT( ) ONLY
            ((UNIQUE) JOB) ((RELATION='EMP')&
             (DNO=50))
```

Example 6: Consider listing all the departments and the average salary of each. This is an example of a query in which a relation needs to be partitioned into groups. A built-in function can then be applied to each group.

```
SEQUEL:
  SELECT DNO,AVG(SAL)
  FROM   EMP
  GROUP  BY DNO
DBC Commands:
  a.  RETRIEVE: ((UNIQUE) DNO) (RELATION='EMP')
```

For each department number 'di' retrieved by (a), the RDBI issues a command:

```
  b.  RETRIEVE: AVG(SAL) ONLY
                (*) ((RELATION='EMP')&(DNO='di'))
```

Example 7: Sometimes it may be desired to partition a relation into groups and then to apply a predicate or a set of predicates which chooses only some of the groups and disqualifies other. These group-qualifying predicates are placed in a special HAVING clause. A predicate in a HAVING clause may compare an aggregate property (e.g., AVG(SAL)) of a group to a constant or to another aggregate property of the same group. The followng SEQUEL statement may be used to list all those departments in which the average employee salary is less than 10,000.

```
SEQUEL:
  SELECT DNO
  FROM   EMP
  GROUP  BY DNO
  HAVING AVG(SAL)<10000
DBC Commands:
  a.  RETRIEVE: ((UNIQUE) DNO) (RELATION='EMP')
```

For each department number 'di' retrieved by (a) the RDBI issues a command:

```
  b.  RETRIEVE: AVG(SAL) ONLY
                (SAL) (RELATION='EMP')&(DNO=
                 'di'))
```

Since the DBC does not make comparisons on aggregate properties, the final selection of DNO based on (AVG(SAL)<10000) is done by software (i.e., by the RDBI) in the front-end computer.

Example 8: Set comparison operators like =, ≠, [IS] [NOT] IN, CONTAINS and DOES NOT CONTAIN are allowed in a HAVING clause as illustrated by this example, which lists the departments which have employees with every possible job title.

```
SEQUEL:
  SELECT DNO
  FROM   EMP
  GROUP  BY DNO
  HAVING SET(JOB)=
         SELECT JOB
         FROM   EMP
DBC Commands:
  a. RETRIEVE: ((UNIQUE) DNO) (RELATION='EMP')
  b. RETRIEVE: ((UNIQUE) JOB) (RELATION='EMP')
               SORT BY JOB
```

For every department number 'di' retrieved by (a), issue the command:

```
  c. RETRIEVE: ((UNIQUE) JOB) ((RELATION='EMP')&
                (DNO='di'))
               SORT BY JOB
```

For each department, the comparison of each of the sets in (c) to the set in (b) is done by software (i.e., by the RDBI).

Example 9: A join operation may be required to return values selected from more than one relation. The names of all employees and the locations where they work may be listed by the query:

```
SEQUEL:
  SELECT EMP.NAME,DEPT.LOC
  FROM   EMP,DEPT
  WHERE  EMP.DNO=DEPT.DNO
DBC Command:
  RETRIEVE: (NAME,DNO) (RELATION='EMP')
            CONNECT ON (DNO,DNO)
            (LOC,DNO) (RELATION='DEPT')
```

Here, there are two attribute lists: (NAME,DNO) for the first query and (LOC,DNO) for the second query. The command is to connect (join) on the two DNO attributes and return as response data triples of the form (NAME,DNO,LOC), where NAME is taken from the first attribute list, LOC is taken from the second list, and DNO is common to both. The RDBI now returns to the user only the pairs (NAME,LOC) by deleting DNO from the triples returned by the DBC.

Example 10: In some circumstances, it is necessary to join a relation with itself according to some criterion. The relation name may then have to be listed more than once and labeled, e.g., X and Y may be two labels for a relation EMP. As an example, the following SEQUEL query will list the employee's name and his manager's name for each employee whose salary exceeds his manager's salary.

```
SEQUEL:
  SELECT X.NAME,Y.NAME
  FROM   EMP X,EMP Y
  WHERE  X.MGR=Y.EMPNO
  AND    X.SAL>Y.SAL
DBC Commands:
  a. RETRIEVE: (MGR) (RELATION='EMP')
               CONNECT ON (MGR,EMPNO)
               (EMPNO) (RELATION='EMP')
```

The only difference between this command and the command for Example 9 is that only one attribute is returned, instead of X.NAME and Y.NAME as well. This is because the AND clause has still got to be considered. Notice that since a manager has at least one employee (in general), a modified command (a') would also have the same effect as (a), yet taking less time to execute. However, (a') is not general enough for all situations.

```
  a'. RETRIEVE: ((UNIQUE) MGR) (RELATION='EMP')
```

For each manager number 'mi' returned by (a), do the following: Send a command

```
  b. RETRIEVE: (NAME,SAL) ((RELATION='EMP')&
               (EMPNO='mi'))
```

and for each (nj,sk) pair returned by (b), send a command

```
  c. RETRIEVE: (NAME) ((RELATION='EMP')&(MGR='mi')
               &(SAL>sk))
```

Notice that the name retrieved by (c) is an employee name, and that returned by (b) is the corresponding manager's name.

Steps (b) and (c) have been written in such a way that for every manager, the DBC accesses all

his employees at the same time. These two steps could otherwise have been written such that for every employee, the DBC accesses all his managers at the same time. But, of course, every employee has a single manager. Therefore, the way we have written the commands is better than its alternative, since fewer number of accesses is required in the former case. The decision is made on the basis of the fact that there are fewer unique values of MGR than there are of EMPNO.

Example 11: SEQUEL permits a label to be used to qualify attribute names outside the block in which the label is defined. The following query uses this feature in listing the suppliers who supply all the parts used by Dept. 50.

```
SEQUEL:
  SELECT SUPPLIER
  FROM   SUPPLY X
  WHERE
         (SELECT PART
          FROM   SUPPLY
          WHERE  SUPPLIER=X.SUPPLIER)
  CONTAINS
         (SELECT PART
          FROM   USAGE
          WHERE  DNO=50)
DBC Command:
  a. RETRIEVE: ((UNIQUE) SUPPLIER) (RELATION=
     'SUPPLY')
  b. RETRIEVE: (PART) ((RELATION='USAGE')&(DNO=
     50))
```

Since the block after CONTAINS has a comparison involving a constant, it needs to be executed only once. This is done by command (b) given above. For each supplier 'si' retrieved by (a), a DBC command

```
  c. RETRIEVE: (PART) ((RELATION='SUPPLY')&
     (SUPPLIER='si'))
```

is sent, and the sets retrieved by (b) and (c) are compared by software.

The same query could have been made in SEQUEL by means of GROUP BY and the special function SET, as given below:

```
SELECT SUPPLIER
FROM   SUPPLY
GROUP  BY SUPPLIER
HAVING SET(PART) CONTAINS
       SELECT PART
       FROM   USAGE
       WHERE  DNO=50
```

The DBC commands would be the same as before.

A Brief Look at Performance

Because of the parallelism involved in the operations performed by the DBC, it should be intuitively clear that user transactions will run faster on the DBC than on a conventional computer. The speed is further enhanced by the fact that a sequence of software operations can be replaced completely by a single DBC command. For example, in order to find all the records satisfying a conjunct of predicates, a conventional system will first determine (in some manner, e.g., via an index) the eligible records. It will then retrieve these records and compare each of them against the given predicates. In the DBC, on the other hand, not only are all the eligible records retrieved in parallel, but it is also true that this set of retrieved records is exactly the required response set. The reason is simply that records are compared against the given predicates simultaneously with their retrieval, thereby rendering unnecessary any subsequent software refinement of the retrieved set.

In the rest of this section we shall consider for study the mass storage requirement, directory storage requirement and the execution time of queries. Rather than a complete detailed analysis, what we provide here is more of a motivation for understanding the difference in performance between the DBC and conventional computer systems. A detailed analysis is presented in [19] and published in [22].

For every tuple stored in a conventional system, the DBC stores a record in its mass memory. While a stored tuple consists of pointers (at least one for each link [15]) and the values for each column of a relation, a DBC record consists of keywords. Within a DBC record, each keyword is made up of a coded attribute as well as a value. In addition, there are one or two special keywords, such as <RELATION, relation-name>, but there are no pointers. If the average length of a value is about double (or more) the size of a coded attribute (which is quite normal), then the DBC mass storage requirement is usually no more than double that of a conventional system, even if no links are defined on the relation. On the other hand, if there are many links, then the DBC storage requirement can actually be somewhat less than that of a conventional system.

With regard to directory storage, it must be pointed out that in the DBC implementation of a relational database, directories are maintained for the relation names and for one clustering attribute per relation. Since the DBC records are primarily clustered by relation name, the size of each entry in the directory for relation names will be quite small. The clustering attribute chosen for a relation is not one of the original attributes but a totally new one. Based on the clustering images and the clustering links, this clustering attribute is allowed to take on as many values as the number of cylinders required to store the entire relation. This is because individual cylinders are content-addressable; so there is no need to keep track of records of a relation within individual cylinders. Furthermore, due to the large size of the cylinders only a few of them will be used for accommodating a relation. For every record of a relation, then, a value is computed (based on its original attribute that appears in the clustering image or link) for the clustering attribute. The record is then stored "close" to other records of the same relation that have a matching value for the clustering attribute. Since the possible number of values of a clustering attribute is very small, the corresponding directories will also be small.

On the other hand, in a conventional system, a multi-page index is maintained for every image of a relation in the form of a modified B-tree [15] Since index entries address pages which are much smaller than cylinders in size, there will be many more index entries per image than the value entries for the DBC clustering attribute. It has been estimated in our analysis [19] that even if there is only one image per relation, directory storage

requirement in this system (for usual relations, say, consisting of 1,000 to 100,000 tuples, each of size 50 to 1,000 bytes) is 10 to 100 times the amount required by the DBC.

Query execution time is normally very much (about 10 to 100 times) faster on the DBC. The reasons are the following: (1) In one secondary storage access, the DBC can content-search an entire cylinder instead of scanning only a single page. Since a normal page size is close to 4,000 bytes, while a cylinder can accommodate as many as 400,000 bytes, it is not unreasonable to expect one or two orders of magnitude increase in speed when the DBC is used; (2) The records retrieved by the DBC are normally the records required in the response set of the query. This compares with the fact that in a conventional system, many of the tuples within a retrieved page will not be immediately required and will, therefore, be wasted; (3) The clustering policy used in the DBC implementation, which we have not discussed in detail, tries to optimize the search policy, without incurring an inordinately large storage overhead.

Concluding Remarks

In the limited space available for this paper, it was not possible to discuss a complete DBC implementation of the relational data model. We have tried to describe mainly the database representation problem and the qeury translation aspects. Details on the record clustering problem and its relation to the clustering links and clustering images have not been included. The data control facilities of System R may be implemented on the DBC in a conventional manner. Taking advantage of the hardware security mechanisms of the DBC, however, an extra degree of flexibility can be attained in solving the security problem [19]. Finally, listed below are the results of a performance analysis [19], which we have broadly overviewed in our last section:

(1) For a relational database, the mass memory of the DBC requires typically up to two times more storage than a conventional system.
(2) The storage used within the DBC structure memory is typically one or two orders of magnitude less in size than that required for storing indexes in a conventional system.
(3) The execution time required for the common SEQUEL queries (simple one-relation or two-relation queries) is normally one or two orders of magnitude faster when the DBC is used.

References

[1] Baum, R.I. and Hsiao, D.K., "Database Computers - A Step Towards Data Utilities," IEEE Trans. on Computers, C25, 12, Dec. 1976, pp. 1254-1259.

[2] Hsiao, D.K., "Data Base Computer - Why and How" Data Base Engineering, IEEE Computer Society, 1,2, June 1977, pp. 4-7.

[3] Lowenthal, E.I., "A Survey: The Application of Data Base Management Computers in Distributed Systems," Proc. Third Int. Conf. on Very Large Data Bases, ACM, New York, 1977, pp. 85-92.

[4] Hsiao, D. K. and Madnick, S. E., "Database Machine Architecture in the Context of Information Technology Evolution," Proc. Third Int. Conf. on Very Large Data Bases, ACM, New York, 1977, pp. 63-84.

[5] Hsiao, D. K. and Kannan, K., "The Architecture of a Database Computer - A Summary," Proc. Third Workshop on Computer Architecture for Non-Numerical Processing, Syracuse, New York, May 17-18, 1977.

[6] Ozkarahan, E. A., Schuster, S. A. and Sevcik, K. C., "Performance Evaluation of a Relational Associative Processor," ACM Trans. on Database Systems, 2, 2, June 1977, pp. 175-195.

[7] Lin, C. S., Smith D. C. P. and Smith J., "The Design of a Rotating Associative Array Memory for a Relational Database Management Application," ACM Trans. on Database Systems, 1, 1, March 1976, pp. 53-65.

[8] Copeland, G. P., Lipovsky, G. J. and Su, S. Y. W., "The Architecture of CASSM" A Cellular System for Non-Numeric Processing," Proc. First Annual Symp. on Computer Architecture, Dec. 1973, pp. 121-128.

[9] Hsiao, D. K., Kannan, K. and Kerr, D. S., "Structural Memory Designs for a Database Computer," Proc. Nat. ACM Conf., ACM, New York, 1977.

[10] Baum, R. I., Hsiao, D. K. and Kannan, K., "The Architecture of a Database Computer - Part I: Concepts and Capabilities," The Ohio State University, Tech. Rep. No. OSU-CISRC-TR-76-1, Sept. 1976.

[11] Hsiao, D. K. and Kannan, K., "The Architecture of a Database Computer - Part II: The Design of Structure Memory and Related Processors," The Ohio State University, Tech. Rep. No. OSU-CISRC-TR-76-2, Oct. 1976.

[12] Hsiao, D. K. and Kannan, K., "The Architecture of a Database Computer - Part III: The Design of the Mass Memory and Its Related Components," The Ohio State University, Tech. Rep. No. OSU-CISRC-TR-76-3, Dec. 1976.

[13] Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," Comm. of the ACM, 13, 6, June 1970, pp. 377-387.

[14] Codd, E. F., "Further Normalization of the Data Base Relational Model," in Courant Computer Science Symp. 6: Data Base Systems, Prentice-Hall, Englewood Cliffs, N.J., May 1971, pp. 65-98.

[15] Astrahan, M. M., et al., "System R: Relational Approach to Database Management," ACM Trans. on Database Systems, 1, 2, June 1976, pp. 97-137.

[16] Chamberlin, D. D., et al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control," IBM Rep. No. RJ1798 (#26096), IBM Thomas J. Watson Research Center, N. Y., June 1976.

[17] Hsiao, D. K., Kerr, D. S. and Ng, F. K., "DBC Software Requirements for Supporting Hierarchical Databases," The Ohio State University, Tech. Rep. No. OSU-CISRC-TR-77-1, April 1977.

[18] Banerjee, J., Hsiao, D. K. and Kerr, D. S., "DBC Software Requirements for Supporting Network Databases," The Ohio State University, Tech. Rep. No. OSU-CISRC-TR-77-4, June 1977.

[19] Banerjee, J. and Hsiao, D. K., "DBC Software Requirements for Supporting Relational Databases," The Ohio State University, Tech. Rep. No. OSU-CISRC-TR-77-7, Nov. 1977.

[20] Banerjee, J., Hsiao, D. K. and Ng, F. K., "Data Network - A Computer Network of General-Purpose Front-End Computers and Special-Purpose Back-End Database Machines," Symp. on Computer Network Protocols, Liege, Belgium, 13-15, Feb. 1978.

[21] Date, C. J., An Introduction to Database Systems, Second Ed., Addition-Wesley, Reading Massachusetts, 1977.

[22] Banerjee, J. and Hsiao, D. K., "Performance Study of a Database Machine in Supporting Relational Databases," Accepted for publication in the Proceedings of the 4th International Conference on Very Large Databases, Berlin, Federal Republic of Germany, Sept. 1978.