ICOT Technical Report: TR-259

TR-259

Performance Evaluation of Superimposed Code Scheme for Relational Operations

by
M. Wada, H. Yamazaki, S. Yamashita,
N. Miyazaki (Oki), Y. Morita and H. Itoh

May, 1987

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Performance Evaluation of Superimposed Code Scheme for Relational Operations

Mitsunori Wada*, Haruaki Yamazaki*, Shouji Yamashita*
Nobuyoshi Miyazaki*, Yukihiro Morita** and Hidenori Itoh**

* Oki Electric Industry Co., Ltd.
** Institute for New Generation Computer Technology

Abstract

This paper describes a method for relational algebraic operations used in *Knowledge Base Engine (KBE)*, which is a hardware attached to a knowledge base machine, *PHi*. PHi stores large numbers of facts as relations and translates queries expressed in Horn clauses to a sequence of relational algebraic commands. Then the relational database management system within PHi executes the commands. The major task of KBE is comparing various tuples. KBE gives a relation an index made of superimposed codes in order to execute algebraic operations effectively. A method for relational algebraic operations using superimposed code is presented, and the performance evaluated.

1. Introduction

To realize high-level knowledge processing, it is necessary to develop a technology to manage a large amount of shared knowledge. In the Fifth Generation Computer Systems (FGCS) project, an experimental distributed knowledge base system, PHI, is being developed, in which some knowledge base systems are connected by a network [Itoh 86]. The system can provide a user interface that seems to be a single knowledge base system logically.

Each site of PHI system contains a relational database management system (RDBMS) as a kernel, which manages knowledge as relations. The RDBMS has special purpose hardware called knowledge base engine (KBE), to execute relational algebraic commands.

This paper describes the performance evaluation of the superimposed code scheme for relational operations as well as how to realize relational algebra in KBE. Section 2 gives an outline of PHI system and KBE. Section 3 explains processing of retrieval in KBE. Section 4 gives an evaluation of this method from the viewpoint of processing steps and disk read time.

2. PHI system

PHI system consists of one or more knowledge base machine PHi's linked by Ethernet-based LAN. The system and a host inference machine (PSI) are also linked by the LAN. We introduced a Horn clause interface between PHI and host machine

or among PHi machines. PHi machine is implemented on PSI, in which KBE is optional. KBE is special hardware to execute relational algebraic commands for knowledge information processing quickly.

2.1. PHi machine

The knowledge base control and management technology is being developed incrementally, based on the database management technology. In this approach, PHi machine has a deductive database in its kernel, which is a relational database management system with a deductive mechanism. The software of PHi machine consists of the knowledge base manager and the distributed environment manager, which links knowledge base managers and provides a user interface that appears to be a single knowledge base system. The knowledge base manager has two layers, a knowledge management layer and a relational database management layer. The relational database management layer is a relational database management system, and the knowledge management layer has a deductive mechanism. The knowledge base manager provides the Horn clause interface (Figure 1).

The knowledge management layer deduces a query represented in the form of a Horn clause, and translates it into relational algebraic commands that can be executed in the relational database management layer, where the optional KBE can execute these commands effectively.

2.2. Knowledge Base Engine (KBE)

In PHi machine, it is assumed that about tens of thousands of tuples will be stored as a relation. In the relational database management layer, the algebraic commands should be executed quickly. The knowledge management layer produces algebraic command sequences comparing tuples or attributes in tuples, and sends them to the relational database management layer. Because of a recursive query, command sequences may contain retrieval operation, set operation, and unique processing.

In KBE each relation has an index table. The index is used to reduce the number of tuples to be processed in later steps. An index table is a set of index records, each of which corresponds to a tuple. An index record has two entries. One entry stores an index value called a superimposed code word (SCW). The other entry stores a pointer to the corresponding tuple. The index value is derived from the corresponding tuple's key attributes.

KBE executes an algebraic command in two phases. The first phase extracts candidates from a relation. The second phase processes commands for these tuples.

Figure 2 presents a logical structure of KBE. The controller is a processor that controls the behavior of KBE. The accelerator is dedicated hardware that compares index values quickly.

3. Retrieval using superimposed code

This chapter explains how to make an index, demonstrates retrieval with it, and discusses the optimum index parameters.

3.1. Index creation

To retrieve large amounts of text data, a method using the superimposed code was proposed [Roberts 79]. This method assigns index records with superimposed code words (SCWs) to a record file that has multiple key words. The method can provide powerful partial match retrieval.

For quick tuple retrieval, an index with SCW was introduced in KBE. An index value is derived as follows:

Suppose a relation, R, consists of N tuples; $R = \{T_1, T_2, ..., T_N\}$. The relation R has r^R key attributes. To compute an index value, a hash function is defined to map attribute values to codes called binary code words (bcw) according to data type. For a given tuple Ti the index value is produced as follows:

(1) The value of each key's attribute of Ti is transformed to the bcw.

(2) All bcws derived are ORed together.

The result of the OR operation is a superimposed code word, an index of Ti. The index table of R is produced by pairing index values and a pointer to the corresponding tuple for all tuples of R. Figure 3 is an example of the index table.

3.2. Retrieval using an index

As explained in 2.2., an index table is used for retrieval. In retrieval query q , r^q key attribute values are specified as a retrieval condition. The first phase, PHASE1, extracts candidate tuples that satisfy the retrieval condition comparing index values with a binary value, called a query mask. The PHASE2 examines the contents of each candidate to take out the tuples satisfying the retrieval condition.

In more detail, query mask, Q, is computed from the set, S_q, of atribute values specified in the retrieval condition, analogous to the way in which index value is derived. First, hash function maps each element in S_q into a bcw. The query mask is derived from OR'ing together the r^q bcws.

If Q and any index value Si doesn't satisfy the following expression (*), the tuple corresponding to the index record can not satisfy the query.

$$Q \wedge Si = Q \cdots\cdots (*)$$

As the result of the PHASE1, a collection π of tuples whose index value Si satisfy the expression (*) is extracted. The PHASE2 examines π so that the tuples which satisfy the retrieval condition can be picked up.

3.3. Design of index parameters

An index value is assumed to have uniform distribution. Then

p(drops) which is defined by equation (#):

p(drops)

= (the number of π elements / the number of all tuples in R) ····· (#)

$$= \sum_{x=0}^{b} \emptyset(b, k, r^q, x)\, p(b, k, r^R, x)$$

$$\emptyset(b, k, r^q, x) = (-1)^x {}_bC_x \sum_{i=0}^{x} (-1)^i {}_xC_i \left({}_iC_k / {}_bC_k\right)^{r^q}$$

$$p(b, k, r^R, x) = \sum_{i=0}^{x} (-1)^i {}_xC_i \left({}_{b-i}C_k / {}_bC_k\right)^{r^R}$$

p(drops) can be computed using parameters b, k, r^R, r^q [Roberts 79], where r^R is the number of key attributes in the relation, r^q is key attributes specified at retrieval condition, b is the length of bcw and k is the weight of bcw. This section discusses optimal value of k. b and k are parameters that specify index nature.

As shown in Figure 4, when b and r^R are fixed the transition of p(drops) according to k's value is examined. The result is that the more r^q increases, the more p(drops) decreases. Hence, the more the number of key attributes specified in a query, the fewer tuples that satisfy the query. The number of tuples satisfying the query is maximal when r^q=1. Thus, the optimization of value k is most important design criterion. As can be seen in Figure 4, this optimal value of k does not change in other curves.

Next, as shown in Figure 5, the transition of p(drops) in the case of r^q=1 according to k's value when b is fixed is examined.

The result is that p(drops) increases in proportion to r^R. Hence, if the number of key attributes specified in a retrieval condition is increased, the number of tuples which do not satisfy queries is expected to grow.

Next, the value of b is set in proportion to r^R. Figure 6 illustrates the transition of p(drops) in the case of $b=r^R*16, 24, 32$ and $r^R=2, 8$. The value of k that minimizes p(drops) is in this range. It is clear that if r^q is a constant, then the transition is very small, within

$$^1/_2 * (b/r^R) \leq k \leq {}^2/_3 * (b/r^R).$$

Therefore, k should be set in this range.

4. Estimated processing time

This chapter estimates the cost of retrieval processing in KBE, and compares it with the costs using the hash table or B^+tree. In KBE, the retrieval operation is the operation executed most often. The relational algebraic command sequence is optimized by selection-first strategy in PHi machine. Therefore, the execution cost of retrieval command affects the entire execution cost of PHi machine.

Here, a retrieval is executed to select all tuples satisfying a retrieval condition, and in the condition, some attribute values are specified and ANDed. When a retrieval command is applied to a relation that has a hash table or B^+tree index of

key attributes, it is done in two phases as in the SCW method. The first phase, PHASE1, selects tuples that match one key value specified in the retrieval condition, called candidate tuples. The second phase, PHASE2, tests the candidates and determines tuples satisfying the retrieval condition completely.

4.1. Comparison time

This section estimates the comparison time to select candidate tuples, and evaluates the effect on retrieval. All index records must be tested in the SCW method regardless of the number of tuples in a relation.

When retrieval is performed for N tuple relation, the PHASE1 should test index records N times. The number of comparison steps to select all candidates is N. When retrieval is performed for the relation that has hash tables, only one access after hashing is reqired to select all candidates. One comparison step is reqired to select all candidates. For the B'tree, the number of comparison steps to select all candidates is Constant * $\log_B N$ (B = number of branches).

Therefore, the number of comparison steps of each method is estimated.

- SCW index method : N
- Hash table method : 1
- B'tree index method : Constant * $\log_B N$ (B=number of branches)

Obviously, the SCW method gives the largest number of steps.

The following values are assumed to estimate the order of comparison time.

Parameters

Number of tuples (N) : 2^{16}
Frequency of comparing one index value (f): 1 ~ 10
Comparison time : 10 ~ 10^2 nsec

Here, f is set to be 1 to 10, since the limitations of the register size of the accelerator forces to compare an index value several time occasionally. The order of each method is

O(SCW index method) = 10^{-1} ~ 10 msec
O(hash table method) = 10 ~ 10^3 nsec
O(B*tree index method) = 10^{-1} ~ 10 μsec .

The total comparison time with the hash table or B*tree is negligible to the disk access time which is in the order of milliseconds. Because total comparison time with SCW index is larger than other method, an accelerator is used to realize the high-speed index processing in KBE.

4.2. Estimated disk access time

This section estimates the time to read candidate tuples from disk hardware in the PHASE2.

The following parameters are used:

s : Average seek time (msec)
r : Rotational latency (msec)
d : Disk transfer time (msec)
P : Page size (Byte)
N : Number of tuples in a relation

n : Number of tuples satisfying a query

In the SCW method, when p(drops) « 1, the same number of pages as candidate tuples will be read. Candidate reading time T_{tuple} for disk access is estimated as follows:

$$T_{tuple} = N * p(drops) * (s + r + P/d)$$

In the expression, $N * p(drops) = N_d$ is the expected number of candidates in the PHASE1.

Figure 7 shows the disk read time. In this figure, the axis of the abscissa is the rate of n to N. Here, the following values are assumed:

- Disk
  - s + r = 25 msec
  - d = 1 KBytes/msec
  - P = 4 KBytes
- Number of tuples in a relation : 2^{16}
- Index value length : 16, 24, 32 bits per attribute

For b/r^R = 24, 32, ... the disk access time to read candidates is represented by curve L_0. For b/r^R = 16 and r^q = 1, disk access time is constant up to a certain point as shown in L_1 and L_2. For $r^q \geq 2$, disk read time is represented by curve L_0.

When tuples are retrieved with the hash table or the B*tree index, candidate tuples are selected by using them for a single attribute. Next, the candidates are tested to see whether they satisfy the query. Candidate hit ratio PHI_T is defined as follows.

PHI_T = (number of tuple satisfying query / number of candidate tuples)

With PHI_T, the expected reading time, T_{tuple}, for candidates can be calculated as follows:

$$T_{tuples} = n * (s + r + P/d) / PHI_T.$$

According to this expression, Figure 7 explains the disk access time for PHI_T = 1/1, 1/2, 1/4, 1/8, 1/16.

In SCW method, for $r^q \geq 2$ the disk access time follows L_0. In other methods, PHI_T is expected to decrease as r^q increases. If the number of the tuples satisfying a query exceeds a certain threshold then the SCW method is better than the other methods in terms of disk access time to read candidates. As PHI_T decreases, the threshold value decreases. In other words, the performance of retrieval by SCW method is expected to be better than other methods as the number of key attributes specified in the retrieval condition increases.

5. Conclusion

This paper described an outline of KBE and a method of relational algebraic operations. KBE is special hardware for executing operations in PHi machine. An index scheme based on superimposed codes is adopted, with which the rapid execution for those operations is performed effectively. An index table is a collection of index records, and one index record corresponds to one tuple. The index record has an index value of one superimposed code word. The following constraints should

be satisfied for designing the index scheme.

(1) The length of the index value b should be set in proportion to the number of key attributes r^R in the corresponding relation ($b = b' * r^R$).

(2) The weight of bcw k for the index value p(drops) should be set in the range of

$$1/2 * (b/r^R) \leq k \leq 2/3 * (b/r^R)$$

to optimize the performance.

The cost of the index comparison time in our method is higher than other schemes, (hash tables, B*tree). However, the retrieval is expected to execute more effectively if the number of key attributes specified in the retrieval condition is larger.

The parallel processing approach may solve the comparison time problem. This wil be a topic for further research.

References

[Itoh 86] Itoh, H.: Research and Development on Knowledge Base Systems at ICOT, Proc. of 12th VLDB, (Aug. 1986), pp. 437-445

[Roberts 79] Roberts, C. S.: Partial-Match Retrieval via the Method of Superimposed Codes. Proc. IEEE, Vol. 67, No. 12(Dec. 1979), pp. 1624-1642

Acknowledgements

The authors are grateful to the members of Third Research Laboratory in ICOT for helpful advice.

Appendix Execution schemes of other operations

This appendix describes how to realize relational algebraic operations in KBE. These operations are frequently used to process a recursive query.

1) Unique operation

Here, the operation that deletes duplicated tuples in a relation is considered. The operation is executed in two phases. The first phase divids the ralation into several groups so that duplicate tuples exist only within the same group. The second phase compares tuples in each group, and deletes duplicates.

Figure A-1 illustrates the grouping of a relation. Here, tuples are grouped so that the tuples that have the same index value are in the same group. Because two tuples that have different index values are different, it is enough to compare tuples in each group.

2) Set operation and implication check

It is necessary to detect duplicated tuples between two

relations when set operations or an implication check between relations are to be executed. The operations are executed in two phases. The first phase is common for these operations. The first phase groups tuples according to the corresponding index value, as in the unique operation. Next, two groups that have the same index value are paired as illustrated in Figure A-2. To detect duplicates in two relations, it is enough to compare tuples between paired groups.

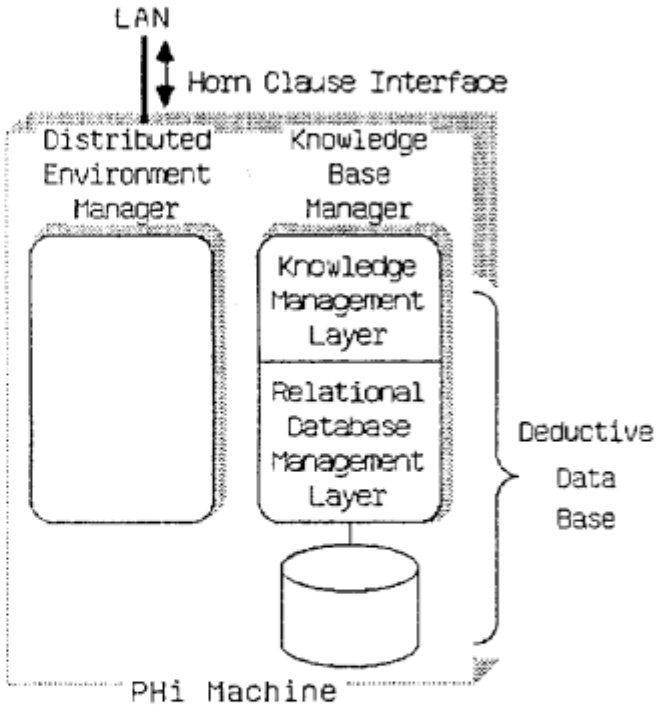

Figure 1 Knowledge Base Machine PHi

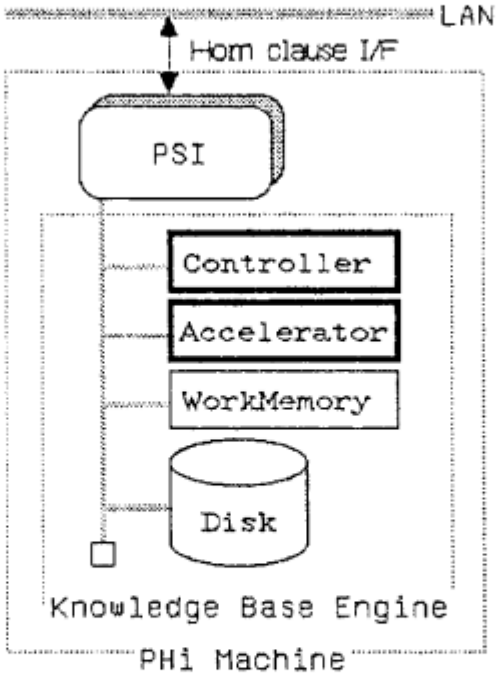

Figure 2 Knowledge Base Engine

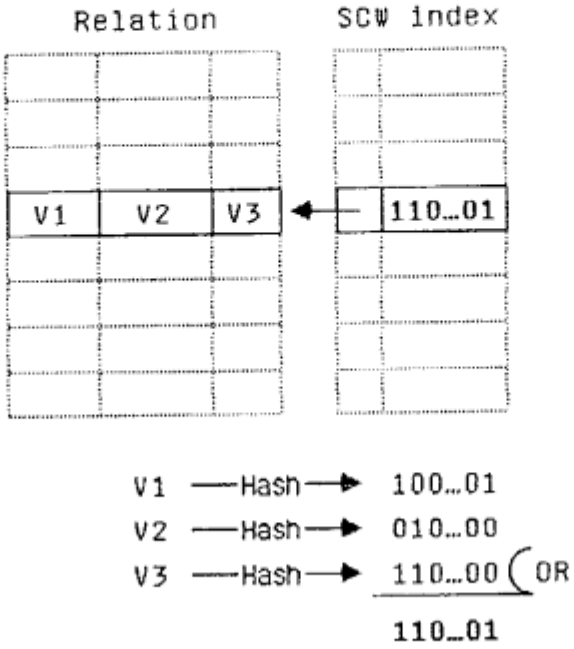

Figure 3 Index Creation

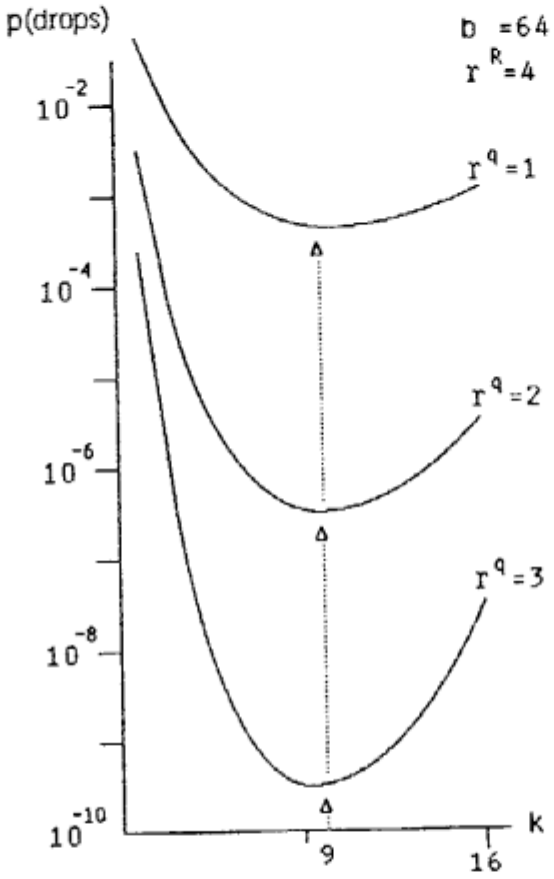

Figure 4 Transition of Selectivity

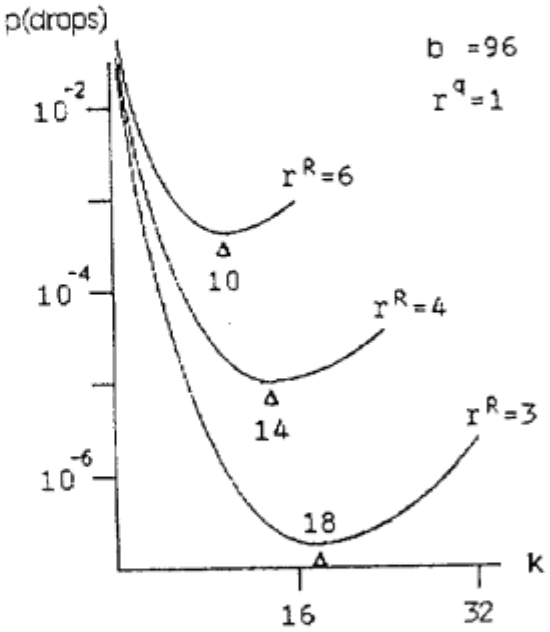

Figure 5 Transition of Selectivity (2)

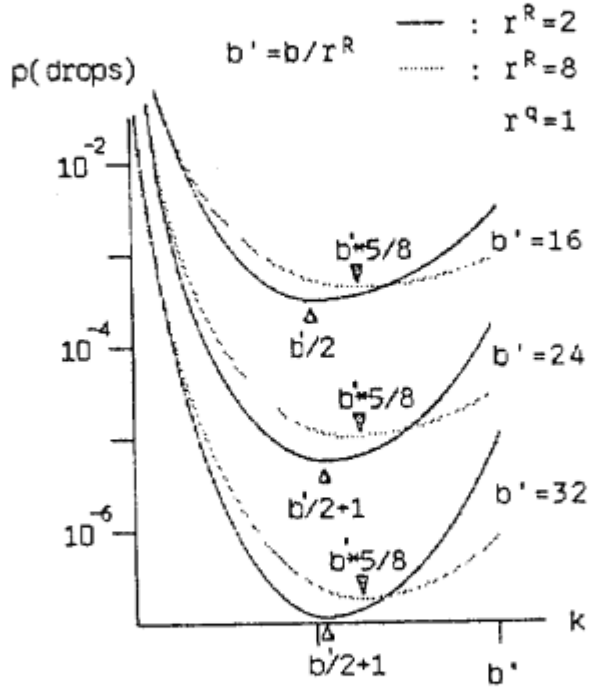

Figure 6 Transition of Selectivity (3)

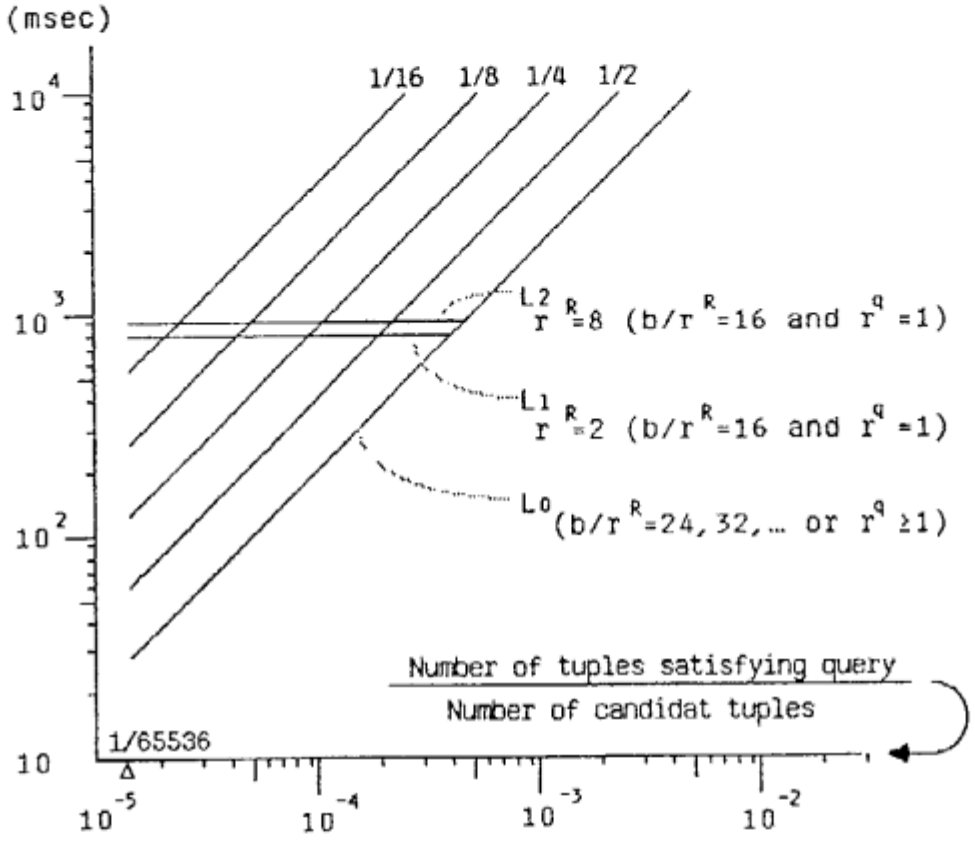

Figure 7 disk access time

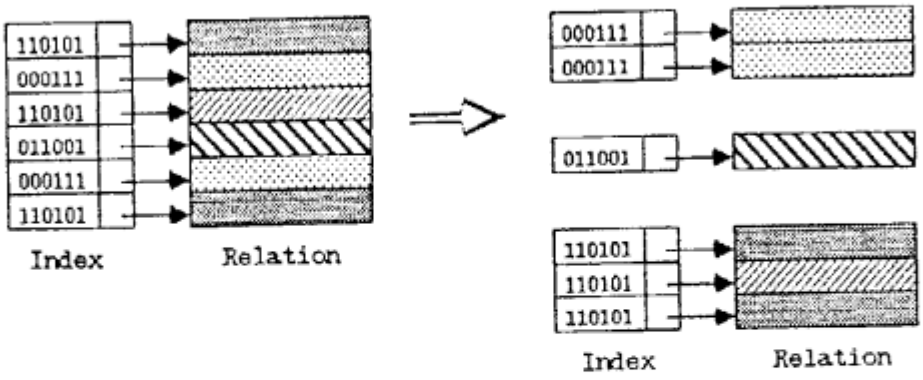

Figure A-1 Grouping by Index Value

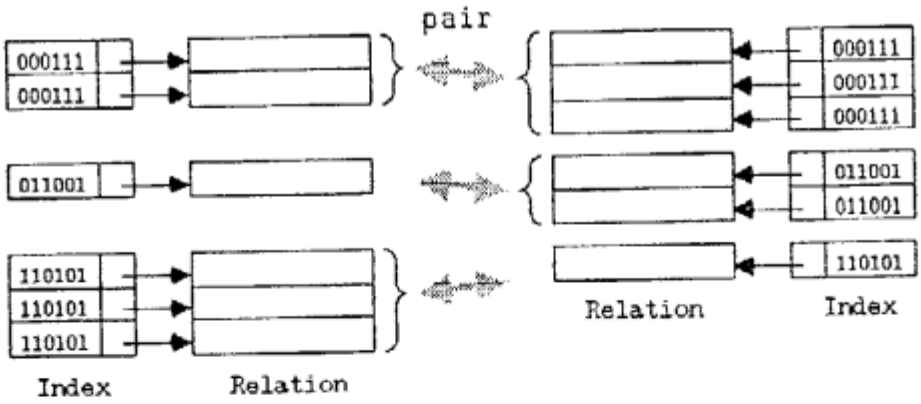

Figure A-2 Pairing for Groups by Index Value