A Multiple Layered Signature Database Architecture for Mobile and Internet Environments

Yuping Yang, Mukesh Singhal
Department of Computer and Information Science,
The Ohio State University
Columbus, OH 43210
{yangy,singhal}@cis.ohio-state.edu

May 27, 1997

Abstract

The performance of very large relational databases is one of the major stumble blocks of their deployment in the mobile and Internet environments. Indexing, hashing, and sorting are still main query execution speed up techniques in actual database implementations. Join index and materialized view have been proposed but not widely accepted due to their large storage requirements. Recently, general multi-layer database concept based on concept-hierarchy has been proposed to improve the performance of query execution. However, there are still much to be done to expand this idea into an implementable database architecture. Implementable new query execution speed up techniques that can significantly speed up query execution, especially when there are a large number of users, yet do not demand excessive amount of extra storage space other than database itself, are in demand. This paper proposes a multiple layer database architecture based on signature method in implementable detail, compares this new architecture with existing query execution speed up techniques, and gives a performance analysis of this new architecture.

key words: Relational database, very large database, multi-layer, join, query processing, signature, grouping, performance, mobile, Internet.

1 Introduction

The overwhelming majority of databases today are still relational databases, and despite of over 20 years of research the interests in enhancing the performance of the relational databases are still high [2, 3, 5, 4, 6, 7, 8] and the performances of query executions are still far from satisfactory. Databases used in mobile computing and the Internet environments such as the World Wide Web are likely be used as direct marketing tools to support consumer queries for information about products and services [9]. These databases are likely to be large and demand very high performance to support interactive accesses while a large number of users querying the database simultaneously. Also, Future databases are likely to be much larger than they are today and query performance is critical to future applications of databases in mobile computing and Internet environments.

Join operation is the single most expensive operation in query processing. So far the implementable techniques for speed up join operation in relational databases that have actually been used in the commercial databases are still various versions of hash join, sort merge join, and index assisted join [1]. Materialized view and join index [12], which are essentially pre-computed joins and can save processing time when used to answer queries, are not widely adopted in commercial databases because of their large extra storage requirements. Recently, multiple layer database has been proposed by Han et al. [18]. This idea uses concept hierarchy as a guideline to extract summary information from a group of data values to form higher layer relations. During a query execution, the summary information in higher layer relations is searched first. Then, the search results from the higher relations are projected to the original database at layer 0 to be searched again. The search results from the higher layer relations can reduce the search space in the original database to speed up the query execution. However, there is still much to do to expand this idea into an implementable database architecture.

The challenges of designing a physically implementable multi-layer relational database architecture are:

- Only the original database at layer 0 contains information that interests user, all relations in other layers only assist in answering queries. Therefore, relations in all layers except layer 0 must be fairly small compared to the original database. This demands a space efficient way of storing the summary information.

- Since the join operation is the most expensive operation in query execution in terms of the amount of disk I/O and the related processing, it is highly desirable that joins, not only selections, can be first executed among higher layer relations to achieve maximum search space reduction in query execution in the original database. However, in previously proposed multi-layer database model as in [18], joins are avoided in higher layer relations due to the intrigue difficulties in carrying them out in higher layer relations.

- It is desirable to have an efficient mapping mechanism to map addresses of tuples of the higher layer relations to addresses of tuples of the original database relations. Once the search in higher layer relations be done, the search results should be mapped to the addresses of the original database relation without a scan of the whole relation. Without such a mapping mechanism, much of the speed up achieved by using higher layer relations is lost.

This paper proposes a new *multiple layer signature database architecture*, MSDBA, which can be added on top of an existing relational database to speed up query processing. MSDBA achieves these benefits by using stored summary information mostly in the form of signatures, to reduce search space in the database in query processing. The main differences between our approach and Han et al. are that in our approach, there is a complete database in each layer so join operation can be done in upper layer databases, and we use signature as the form of storage for summary information.

2 Background: Signature Method

The term *address* in this paper means physical address, i.e., page number and the location in the page. All entries in a tuple are stored consecutively and a tuple address is the address of the first entry of the tuple. However, all discussions are true when the address means logical address, such as the surrogate [19].

A *signature* is a 0/1 bit pattern obtained by hashing data values. Several signatures can be superimposed to form a group signature. Figure 1 shows a relation with an attribute "profession". Each value in "profession" is hashed into a signature, called an individual signature. Then, some individual signatures are grouped and superimposed to form group signatures, *sig1, sig2*, and *sig3*. Only group signatures are stored in MSDBA.

Consider the following query:

$$\begin{array}{ll} select & R.name \\ from & R \\ where & R.profession = \text{``}secretary\text{''} \end{array}$$

The value "secretary" in the predicate of the query can be hashed into a signature *qsig*, called a query signature. *qsig* is checked against group signatures. A match is defined as each bit of 1 in *qsig* be matched by a bit of 1 in the same position of the group signature.

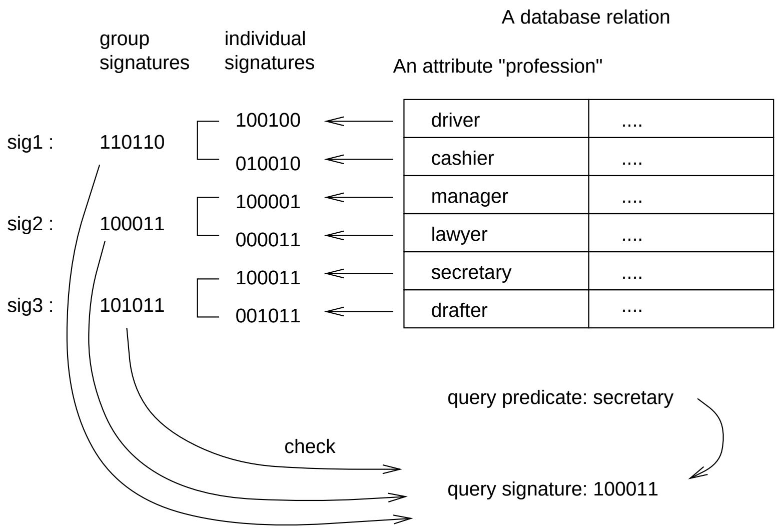

Figure 1: Individual signatures, group signatures and the query signature

The group of values that generate *sig1* cannot possibly contain value "secretary" because *qsig* does not match *sig1*. Although *qsig* matches *sig2*, further checking shows that this is a false match. Only *sig3* has a true match with *qsig*. If the group is large, for example, the average size of a group is 100, and a lot of group signatures are not matched, then a significant amount of search can be reduced.

$$\begin{array}{llll}
secretary & 100011 & \longleftarrow & query \;\; signature \\
sig1 & 110110 & \longleftarrow & not \;\; match \\
sig2 & 100011 & \longleftarrow & false \;\; match \\
sig3 & 101011 & \longleftarrow & true \;\; match
\end{array}$$

The purpose of using signature is mainly to reduce the search space [10]. Signatures are only used for attributes with text data domain, such as names, descriptions, etc. The numerical data such as weight, distance, etc, are not hashed to signatures because numerical values are often used in predicates of range queries.

3 Multiple Layer Signature Database Architecture

To speed up query processing, MSDBA contains one or several small signature databases on top of a relational database, called the *main database*. An MSDBA together with the main database will be called a *multiple layer database,* MSDB.

Query executions are first done in the signature database. This gives approximate locations of interesting values in the main database and will be done very fast since the signature database is much smaller than the main database. After that, the results are mapped to the main database and query executions are done in the main database. Query execution in the main database will be done very fast since the search in the signature database has already revealed the approximate and possible locations of all interesting data.

Three challenging problems need to be addressed. First is how to design signature databases so that they are small yet provide accurate information about locations of relevant data in the main database. Second is how to organize signatures so that relational operations, especially join, can be performed efficiently on them. Third, once the search is done on the signature database, how to map the results to the main database to reduce search space in the main database. The first problem is solved by using group signatures, the second by organizing group signatures into a relational database like structure, i.e., the signature database, and the third, is addressed by providing the *addressing tables*. Next, we explain these concepts in detail.

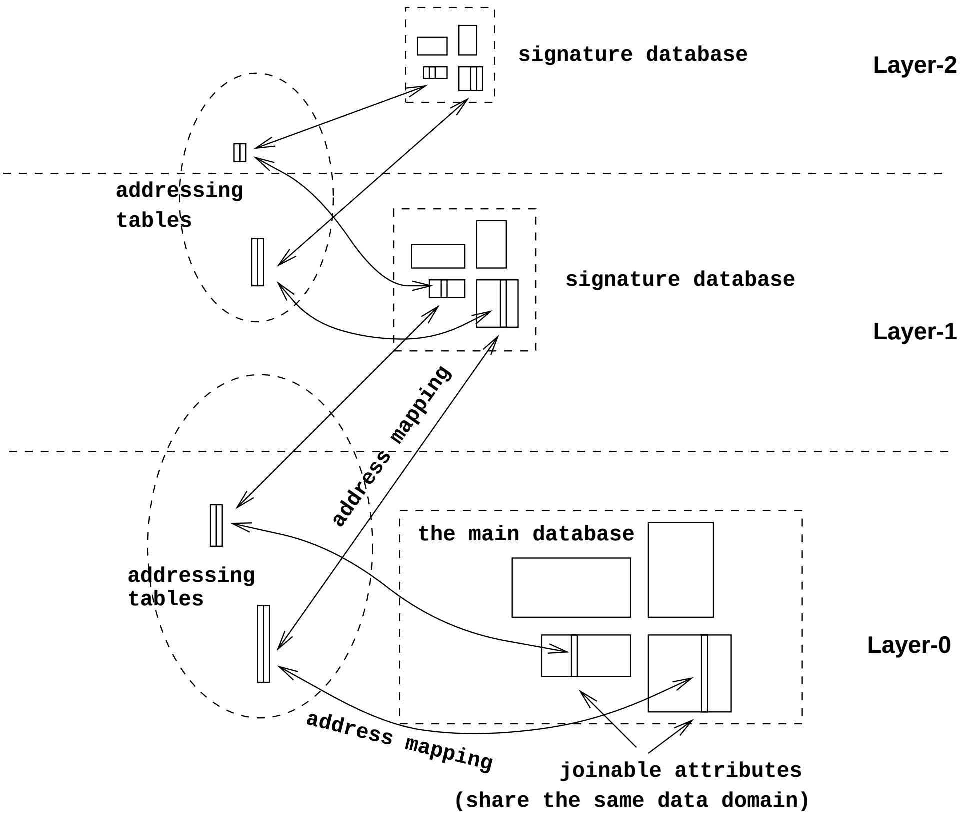

Figure 2: The architecture of an MSDB

Signature databases: For each joinable attribute[1] in each relation of the main database, a *signature relation* is constructed that becomes a part of a signature database. The signa-

[1]A joinable attribute in a relation is an attribute that shares the same data domain with another attribute in another relation, and therefore join between these two attributes is possible

ture relation has the same set of attributes as the corresponding main database relation. A tuple in a signature relation corresponds to a group of tuples in the main database relation. Each entry in a tuple of a signature relation contains a group signature if the corresponding attribute has a text data domain. Treatments of numerical values will be discussed in Section 6. The grouping of tuples in the main database relation is based on the grouping of the values in the joinable attribute of the main database relation.

If on average a group signature is generated from 100 text values in the main database, and each value in the main database takes approximately the same space as a signature, then the size of a signature relation is about 1% of the corresponding main database relation. If on average there are three joinable attributes per relation in the main database, then the total size of the signature database will be about 3% of the main database.

A schematic diagram of MSDB is shown in Figure 2. MSDB is divided into layers. Layer-0 consists of the main database. Layer-1 consists of signature databases whose signature relations are derived from the main database. The group signatures in a signature relation at layer-1 can be further grouped and superimposed to form an even higher layer group signatures at layer-2. The small signature database can be duplicated in each layer to allow parallel processing of queries in each layer before searching the lower layer database. Search in higher layer signature database is always done before the search in lower layer database. The search result from higher layer database can effectively restrict the search space in lower layer database.

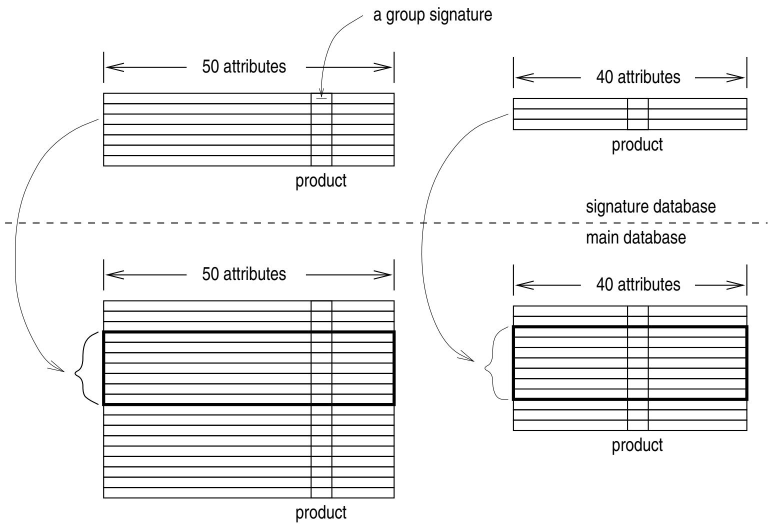

Figure 3: Correspondence between main database relations and signature relations

Let's assume that a database has two relations as shown in Figure 3, one relation *market* has 50 attributes and 100,000 tuples, and the other relation *supplier* has 40 attributes and

60,000 tuples. *Product* is the joinable attribute in these two two relations. The signature database consists signature relations $mark_S$ and sup_S, derived from *market* and *supplier*, respectively. The average size of groups in constructing $mark_S$ is 50, and is 200 for sup_S. So number of tuples in $mark_S$ is 100,000/50 = 2,000 and in sup_S is 60,000/200 = 300. Consider the following query:

$$\begin{array}{ll} select & market.sale\_price,\ supplier.supply\_price \\ from & market, \\ & supplier \\ where & market.product = supplier.product\ \ and \\ & market.category = \text{``}homedevice\text{''}\ \ and \\ & supplier.class = \text{``}electronic\text{''}. \end{array}$$

To make a fair comparison, let's assume that all tuples are stored consecutively in each relation, disk page size is 512 bytes. All addresses and signatures are 4 bytes and text values are 10 bytes long. So each tuple in *market* is $50 \times 10 = 500$ bytes, and in *supplier* is $40 \times 10 = 400$ bytes. Joins are performed using the nested loop method. When performing join the relation *market* or $mark_S$ is always in the outer loop and *supplier* or sup_S is always in the inner loop. Effective memory space is 10 MB and is equally used by pages from the two relations of the join.

Without using signature database, disk I/O to *market* is $(100,000 \times 500)/512 = 97,657$ pages. Half of the available memory, 5 MB, can be used to store pages from *market*. 97,657 pages can be divided into 10 buckets, each each bucket can reside entirely in the memory. So disk I/O to *supplier* is $(10 \times 60,000 \times 400)/512 = 468,750$ pages. Total disk I/O to the main database is 566,407 pages.

When the signature database is used, the join is first performed between attribute *product* in signature relations $mark_S$ and sup_S. Disk I/O to $mark_S$ is $(2,000 \times 4)/512 = 16$. Since 16 pages can reside entirely in the memory, disk I/O to sup_S is $(300 \times 4)/512 = 3$. The match between the signatures in $mark_S$ and sup_S is defined as two signatures are identical. The details that make this match possible will be discussed later. The matched signature tuples in both $mark_S$ and sup_S are then mapped to the main database. If 60 tuples in $mark_S$ matched with 40 tuples in sup_S, then there are $60 \times 50 = 3,000$ tuples in *market* and $40 \times 200 = 8,000$ tuples in *supplier* need to be searched.

In general, we can't assume these tuples be stored consecutively. Therefore, for each tuple at least one page is to be retrieved. Also, some additional disk I/O are needed due to some extra data structure in MSDB. Assuming all disk I/O is 50,000 pages, that is only 8.8% of the disk I/O when the query is directly executed on the main database. More precise performance analysis will be given in Section 7.

Similar comparisons can be made when other techniques such as indices or hash join are applied in query processing. The important point is that the signature database method is orthogonal to other techniques like inverted indices and hash join [20], and can be used in conjunction with all these techniques to enhance the performance of database.

Addressing tables: After searching the signature database, a set of signature tuples are found. These tuples represent the approximate and possible locations of the interesting data in the main database. However, if no special mechanism is provided, the entire main database relation has to be scanned and the benefits of using the signature database may be lost. An addressing table provides such a mechanism to map the signature tuples to the tuples in the main database for further search.

Each entry in the addressing table has two components: the address of an higher layer relation tuple and the set of addresses of tuples in a lower layer relation. The connection between addresses in these two components is such that the group signature in the joinable attribute of the higher layer relation tuple is a group signature obtained by superimposing signatures from the lower layer addresses.

After searching the higher layer signature database, addressing tables are used to map the resulting tuples in the higher layer relation to the corresponding set of the tuples in the lower layer relation.

The size of the addressing table could dominate the size of the MSDBA. If a main database relation has 100,000 tuples and three joinable attributes, then three addressing tables are needed each containing addresses of 100,000 tuples. However, typically, the number of joinable attributes is much less than the total number of attributes in a relation. If this relation has 50 attributes and if each address takes about the same space as a typical text value in the main database relation, then all three addressing tables together take only 6% space of the main database relation.

The addressing tables usually don't need to be duplicated to enhance performance since they are likely to be much less loaded than the database at any layer.

Bonding tables: Each of the signature relations that stems from the same main database relation contains group signatures, and the groupings are done based on values in different joinable attributes. Once the signature relations are constructed, there is no way to find out the connections between these signature relations by directly examining these signature relations, except to go through the grouping process again in the main database relation. Therefore, bonding tables are constructed to solve this problem.

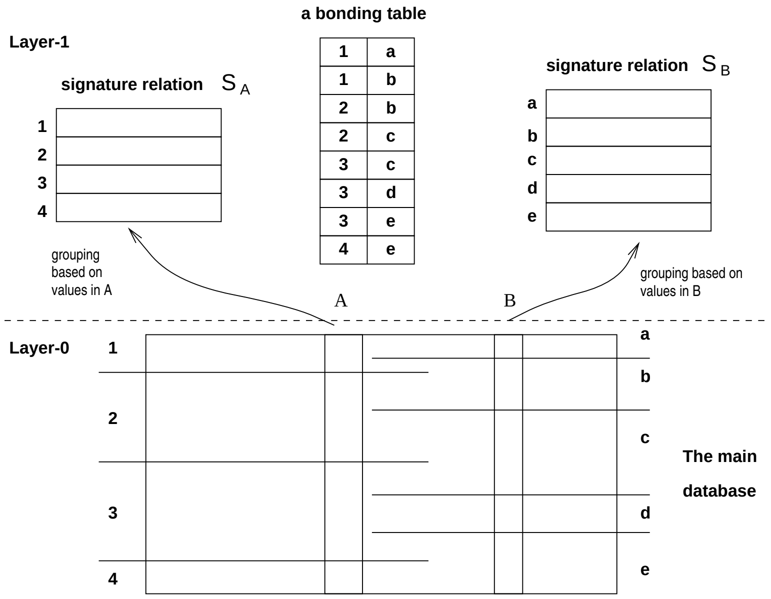

Figure 4: A bonding table between two signature relations

A main database relation may have more than one joinable attributes, so more than one signature relations may be constructed and bonding tables are built between each pair of these signature relations.

Every row of the bonding table has two components each containing an address of the corresponding signature in one of two signature relations, as shown in Figure 4. The horizontal line segments on the main database relation represent groupings of the tuples in the main database relation. Tuple 2 in signature relation S_A corresponds to tuples b and c in signature relation S_B, and this is recorded in the bonding table.

Bonding tables could have more but smaller entries compared to the signature relations. However, the signature relations are much smaller than main database relations, so the size of the bonding table will not become a problem if grouping grouping is done properly.

Optionally, join index [11] can be used between two signature relations that stem from different main database relations. Since the signature relations are much smaller than the main database relations, the join indices in MSDB are much smaller than those generated directly from the main database [11].

Like addressing tables, bonding tables usually don't need to be duplicated since they are likely to be much less loaded than the signature relations.

4 Implementation Considerations

Grouping of values: Tuples in a main database relation are grouped by values of one of its joinable attributes. Let A be a joinable attribute with data domain D in relation R and S_D be the set of all values in all attributes of various main database relations that share the same data domain D. The values in S_D are partitioned into sets. We will give an example to briefly explain this process. Each set will then be given an unique identification number such as a surrogate [19].

Then, from each set in S_D, values that belong to A in R are selected into a group and group signature is constructed. Do the same for each joinable attribute. All group signatures that come from the same set in S_D bear the same surrogate.

Groupings can be done with the help of concept hierarchies. Concept hierarchy [14, 16, 17, 18] is a tree like structure. Each concept is represented as a node and the relationship between a node and its parent node denotes an "is-a" relationship. Concept hierarchies are application specific and can be provided by domain experts.

In the following example, attribute "profession" in R has the same data domain as attribute "job" in S. All values in these two attributes are grouped based on the concept hierarchy shown in Figure 5.

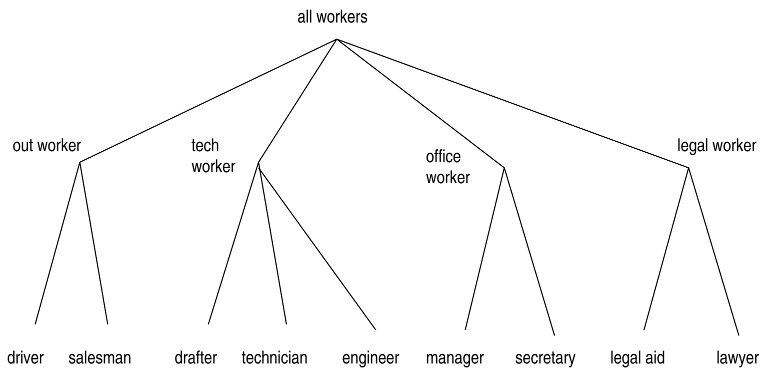

Figure 5: A concept hierarchy used in grouping

The grouping makes four sets out of profession and job, i.e., "out worker", "tech worker", "office worker" and "legal worker". Four group signatures are obtained by taking professions in R from the four sets and four group signatures are obtained by taking jobs in S from the four sets. The group signatures of these two signature relations that come from the same set are identified by the same surrogate. This approach makes the join between signature relations simple.

profession
driver
engineer
drafter
salesman
manager
lawyer

job
driver
secretary
lawyer
legal aid
drafter
manager
technician

$\Longrightarrow$

groups in profession	groups in job	sets
driver salesman	driver	out worker
engineer drafter	technician drafter	tech worker
manager	manager secretary	office worker
lawyer	lawyer legal aid	legal worker

The MSDB architecture can benefit from research in data classification, especially the study of concept hierarchies in data mining [15, 16, 17].

Reducing size of addressing tables: Conceptually, each entry in the addressing table should look like $(address_{high}, address_{low})$, where $address_{high}$ is the address of a tuple in a higher layer relation and $address_{low}$ is the address of a tuple in a lower layer relation. However, only one $address_{high}$ is needed for each group of $address_{low}$ in each entry of the addressing table. Therefore, the bulk of the addressing table is comprised of the addresses of the lower layer relation.

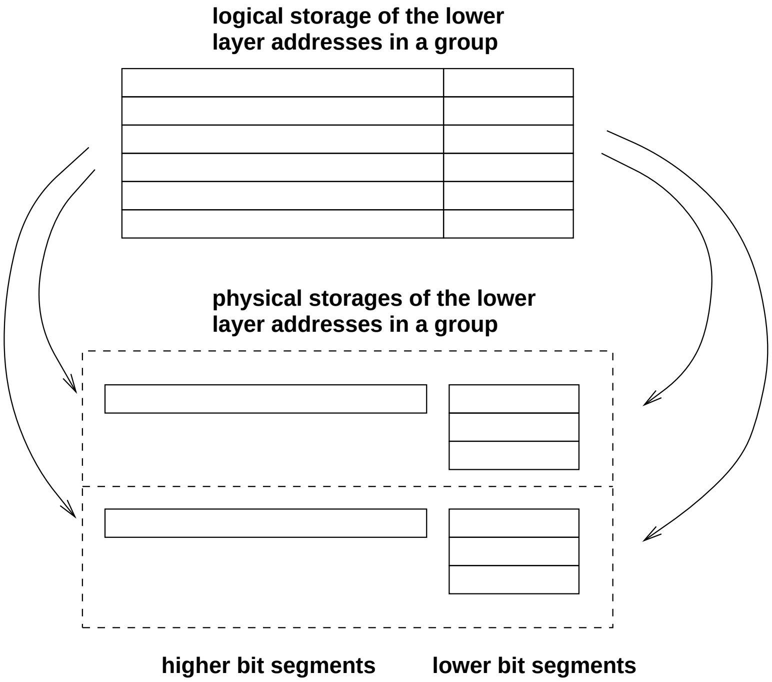

Figure 6: Separation of bits in address to reduce the storage requirements

A lower layer address can be divided into, for example, a higher bit segment and a lower bit segment, as shown in Figure 6. Let's assume the total length of an address is 32 bits, the higher bit segment has 23 bits, and the lower bit segment has 9 bits. Let there be a total of 100 lower layer addresses correspond to a higher layer address. In the group of lower layer addresses, there are 7 distinct higher bit segments and 100 distinct lower bit segments. Then, these segments are partitioned into 7 groups. Each group has a distinct higher bit segment and all its corresponding lower bit segments. On average, there are 14 lower bit segments per group. The total space they occupy is $7 \times 23 + 7 \times 14 \times 9 = 1,043$ bits. Simply storing these lower layer addresses requires $100 \times 32 = 3200$ bits. So using this technique reduces space up to 67%.

To get a feel of how much space the addressing tables take, let's assume that these reduction techniques can reduce the size of addressing tables up to 50%, each address in the main database has four bytes, and on average each data value in the main database has 8 bytes. Also, let's assume on average there are 20 attributes per relation and three of them are joinable attributes. Then the ratio of the total size of the addressing tables compared to the main database is:

$$r = \frac{50\% \times 4 \times 3}{8 \times 20} = 3.75\% \tag{1}$$

Updating MSDBA: Updates to the MSDBA are necessary when there are insertions, deletions and writes to entries of the main database. The construction and update of the MSDBA can be done during the low-load period of the database.

Duplications of signature databases: Duplication of signature database in each layer is done to alleviate the contention for disk heads when there are a large number of queries in the MSDB simultaneously.

The number of the duplications of the signature databases can be determined by the load to the MSDB. A "quick increase, slower decrease" policy may be adopted. When the number of queries are changing slowly, the average number of queries over the past time period t_0 can be computed and used to determine the number of duplications. When there is a sharp increase in the number of queries, the criterion is changed so that the actual number of queries querying the database is used to determine the number of duplications.

5 Comparisons with Existing Techniques

Past efforts in enhancing query execution: The practical and most commonly used techniques to speed up query processings are indexing, sorting, and hashing. Primary index and clustering index can only be used on no more than one attribute. If more than one attributes need to have index, only inverted index can be used, which are very large. Sorting is limited only to one attribute of the relation, and if used for multiple attributes, multiple sortings have to be done at the time of query execution. Hashing requires large amount of memory to be effective. The approach proposed in this paper is orthogonal to all the previous techniques in the sense that MSDBA achieves speed up by reducing the search space.

Next we compare MSDB with the join indices and multi-layer databases since these two techniques bear some similarities in concept to MSDB.

Layered database: Multiple layer database models have been proposed earlier [21, 22, 23]. However, the focus of these models has been database security rather than in query processing.

Han et al. [18] proposed a *multiple layer database,* MLDB. MSDB differs from MLDB in several important ways. In MSDB, there is a clear separation between layers and complete database reside in each layer. Joins cannot be performed in higher layers of MLDB while in MSDB, joins are performed first in higher layers. Finally, signatures are used in MSDB to construct higher layer relations.

Join indices: A join index [11] is a pre-joined relation of arity two. Each tuple in a relation is uniquely identified by a surrogate [19]. Each tuple in a join index contains two surrogates, each points to a tuple in a different relation and these two tuples satisfy the join predicate. A join index could be quite large compared to the two operand relations. Because of the large size of the join index, two identical copies of the join index are usually needed to facilitate search in the join index. Each copy is clustered along one of the columns of the join index. This makes the join index even larger.

A comparison among join index, materialized view and hybrid-hash join by Blakeley et al. [12] found that when the data size is huge and the selectivity is low, the performance of the hybrid-hash join is the worst, the materialized view is better, and the join index is the best. A recent work by Z. Li and K. A. Ross [25] has improved the join index method in that a join operation requires only the following: one sequential pass through each participating relation, one sequential pass through the join index, and two passes through some temporary files whose total size is half that of the join index.

However, a join index is much larger then an MSDBA and the large size of the join index not only limits its use because of excessive amount of additional disk space, but also drags down the performance. In comparison, the MSDBA method has no large structures, such as the join index, to scan, and when the data size is huge and the selectivity is low, the MSDBA method may even require much less than a single pass to each participating relation.

6 Attributes with Numerical Data Domains

Treatment of numerical values in MSDB is similar to that of the text values. Instead of group signatures, statistics like average, minimum, and maximum values are used to represent a group of numerical values in a compact way. Partitioning by range or clustering, such as BIRCH [13], can be used to group numerical values.

For rational numbers, the groups should be such that between any pairs of values within the same group lies no values belonging to other groups. This property is called the "convex" [2] in mathematics and is needed to process range queries such as all products with prices below \$1.2 per piece. Let us use an example to explain range query processing in MSDB. In the following query, the values of attribute *product* are selected based on their relative difference in relation *market* and *supplier*:

$$\begin{array}{ll} select & market.product \\ from & market,\ supplier \\ where & \mid market.price - supplier.price \mid > 0.35 \end{array}$$

Assuming all product prices in *market* and *supplier* are between \$0.5 and \$1.0, and prices in every 0.1 value range, starting from \$0.5, is in a group as shown in Figure 7, then, there are total of 5 price groups in *market* and *supplier*. During a query execution, if some prices in one group in *market* and another group in *supplier* satisfy the query predicate, these two groups are considered match. For example, *market.group_1* is matched with *supplier.group_4* and *supplier.group_5*, but not *supplier.group_3*, since none of the prices in *market.group_1* can match with prices in *supplier.group_3*.

7 Performance Analysis

We use the query given in Section 3 as a typical query to make our comparison. Comparison is made between MSDB with hybrid-hash join and B+ index (both on signature relations and on main database) to just hybrid-hash join and B+ index. Comparisons are done for disk I/O only and the unit of comparison is number of pages.

[2]For data values in 2 dimensional space, same sized and shaped rectangular boxes could be used to provide a convex grouping.

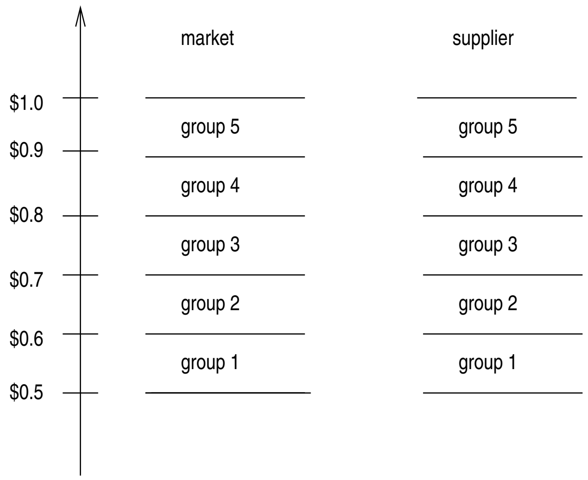

Figure 7: Groups of prices in market and supplier relations

Assumptions Variables and their assumed values used in performance comparison computation are listed in Table 1. There are rationals behind these assumptions as briefly explained in the table. One important check criterion is that v_{ms}, the selection selectivity in the main database should not affected by the group size g, because v_{ms} should only depend on the number of qualified tuples in the main database. This can be checked by substituting v_j into v_{ms}'s expression, and v_s into v_j's expression. Also, v_{ms} should be a lot more smaller than v, because v_{ms} is the selection selectivity after a join operation, and v is not. Here we assume that v_{ms} is 1/10 of v.

Each main database relation has a attributes, t tuples, and each tuple has a bytes, so each main database relation has $t * a * e/p$ pages.

Formula We adopt the formula in [12] for computing IO_{ii}, the number of pages accessed when accessing k_f tuples in a relation having m_f pages and n_f tuples via an inverted B+ index of two or three levels [12]:

$$IO_{ii}(k_f, m_f, n_f) = [Yao(k_f, m_f, n_f) + Yao(k_f, n_f/FO, n_f) + T3(k_f, n_f), \tag{2}$$

where

$$T3(k_f, n_f) = Yao(Yao(k_f, n_f/FO, n_f), n_f/(FO * FO), n_f/FO). \tag{3}$$

The formula $YAO(*)$ is the number disk accesses without any search overhead, and is by Yao, S. B. [24]:

$$Yao(k_f, m_f, n_f) = m_f - m_f * \prod_{i=1}^{k} \frac{n_f - (n_f/m_f) - i + 1}{n_f - i + 1}. \tag{4}$$

t		average number of tuples in each relation
g		group sizes for generating group signatures
a	50	average number of attributes in a relation, assumed
e	4	average number of bytes of a data entry, assumed
i	4	number of bytes in an address, assumed
s	4	number of bytes in a signature, assumed
p	512	number of bytes in a page, assumed
M	20480	number of available pages in memory, 20480 pages takes 10 MB, assumed
FO	$\sqrt{t/100}$	B+ tree fan out. This is obtained from the assumption $t = 100 * FO^2$, i.e., B+ tree has only two levels, assumed
f_b	$\sqrt{g}$	as g increases, bonding table size slowly increases, assumed
d	$1/\sqrt{\sqrt{g}}$	average size reduction ratio for addressing tables d slowly improves as group size g increases, assumed
v	$t/10$	selection selectivity v in main database, assumed
v_s	$(t * \sqrt{g})/(g * 10)$	in MSDB, selection selectivity in signature relation v_s is proportional to the size of the signature relation t/g and $\sqrt{g}$, assumed
v_j	$v_s/10$	in MSDB, subsequent join selectivity v_j in signature relation, assumed
v_{ms}	$v_j * g/\sqrt{g}$	in MSDB, after processing in signature relations, selection selectivity v_{ms} in main database is proportional to signature relation join selectivity v_j be expanded g times, and this selectivity improves as g increases, assumed

Table 1: Variables and values used in performance comparison

Performance of Hybrid-hash join with B+ index: The disk I/O for two direct selection operations in two main database relations is:

$$C_{hs} = 2 * IO_{ii}(v, m, t); \tag{5}$$

where $m = t * a * e/p$ is the number of pages in a main database relation.

Hashing in the hash join operation can be combined with the writing phase of the selection operation so the hashing doesn't cost additional disk I/O. In other words, the writing disk phase of the selection operation actually writes resulting tuples in hash buckets.

After the selection operation, a hybrid-hash join operation needs to be performed on joinable attribute "product". Let

$$B_h = max(\ 0,\ \ \frac{F_h * v * a * e/p - M/2}{M/2}\ ), \tag{6}$$

where F_h is the space overhead factor for hashing [12] in database relations. Tuples in each operand relation are hashed into $B_h + 1$ buckets, each has at most $M * p/(2 * a * e)$ tuples (note that the unit of M is page) so that one bucket from each of the two operand relations can be fit into memory.

Assuming that the pages in each bucket are consecutive and are 50% occupied, this implies that the number of pages in a bucket is M, and the disk I/O for writing pages of all buckets to disk is:

$$C_{hw} = 2 * B_h * (M/2). \tag{7}$$

This is also the number of disk I/O to read in pages to perform join operations:

$$C_{hj} = 2 * B_h * (M/2). \tag{8}$$

Performance of MSDBA with hybrid-hash join and B+ index: Let $n = (t * a * s)/(g * p)$ be the number of pages in a signature relation, then the disk I/O for two selection operations (for attributes $market.category$ = "$homedevice$" and $supplier.class$ = "$electronic$") in the two signature relations is:

$$C_{ss} = 2 * IO_{ii}(v_s, n, \frac{t}{g}); \tag{9}$$

Let $b = (t * f_b * i)/(g * p)$ be the number of pages in a bonding table, then the disk I/O for accessing two bonding tables (to project the selected results from attribute $market.category$ to $market.product$ and from attribute $supplier.class$ to $supplier.product$) is:

$$C_{sb} = 2 * IO_{ii}(v_s, b, \frac{t * f_b}{g}); \tag{10}$$

Let

$$B_s = max(\ 0,\ \ \frac{F_s * v_s * a * s/p - M/2}{M/2}\ ), \tag{11}$$

where F_s is the counterpart of F_h in signature relations, and we assume they have the same value.

Then, similar analysis get the disk I/O for writing out the selection results is:

$$C_{ssw} = 2 * B_s * (M/2). \tag{12}$$

This is also the number of disk I/O to read in pages to perform join operations in the signature relation:

$$C_{ssj} = 2 * B_s * (M/2). \tag{13}$$

Now, all the computations in the signature database have been completed, and the results need to be projected to main database for further query processing. Let $r = (t * i * d)/p$ be the number of pages in an addressing table, the disk I/O for accessing two addressing tables is:

$$C_a = 2 * IO_{ii}(v_j * g, r, t); \tag{14}$$

Now, the query execution has to be repeated again in the main database with reduced search space. The disk I/O for two selection operations in two main database relations is:

$$C_{ms} = 2 * IO_{ii}(v_j * g, m, t) \tag{15}$$

where $m = t * a * e/p$. Let

$$B_m = max(\ 0,\ \ \frac{F_m * v_{ms} * a * s/p - M/2}{M/2}\ ), \tag{16}$$

where F_m is the counterpart of F_h in signature relations, and we assume they have the same value.

Then, similar analysis get the disk I/O for writing out the selection results is:

$$C_{smw} = 2 * B_m * (M/2). \tag{17}$$

This is also the number of disk I/O to read in pages to perform join operations in the main relation:

$$C_{smj} = 2 * B_m * (M/2). \tag{18}$$

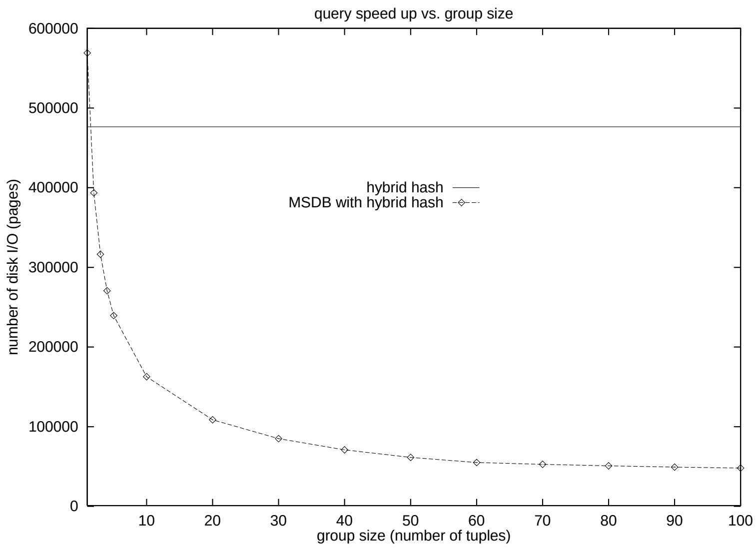

Figure 8: Performances speed up as group size changes

Computation results: Figure 8 gives the results when $t = 1000000$, and group size g varies from 1 to 100. Figure 9 gives the results when the number of tuples t varies from 100,000 to 1,000,000 while group size is fixed at 100. From Figure 8, we can see that when the number of group size increases, the performance of the MSDB improves. The group size critically affects the performance of the MSDB. But the curve of performance improvement levels out as the group size approaches to 100 due to worsen selectivity in signature relations. From Figure 9, we can see that performances of both the hybrid-hash join with B+ index and the MSDB with B+ index linearly decrease as the number of tuples in the database increases; however, the performance of the MSDB with B+ index is much better than that of the hybrid-hash join with B+ index.

8 Concluding Remarks

In this paper a new database architecture, MSDB, was proposed and all its major components and major issues in its implementation were discussed. MSDB uses signature to embody summary information from groups of data values in the database, organizes the signatures into relational database like structure mirroring that of the original database, allows query execution, especially join operation to be first conducted in compact signature database to locate approximate and possible locations of interesting data, thus speed up the over query processing. Because of the small size of the signature database, the throughput of MSDB can also benefit from parallel processings in duplicated signature databases.

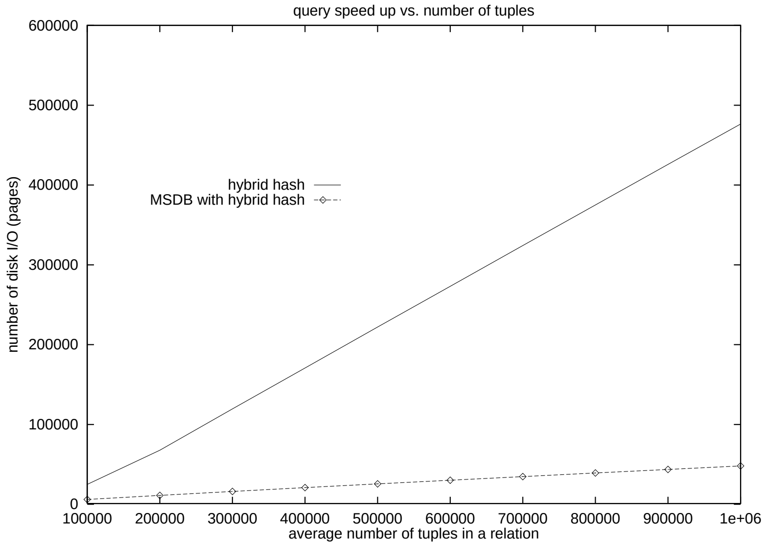

Figure 9: Performances speed up as number of tuples changes

Comparisons between other query processing techniques and MSDB have been made, and the advantages of MSDB are clear from the comparisons.

The main features that makes MSDB attractive in mobile computing environment and Internet environment such as the World Wide Web, as well as in very large databases are:

1. its performance scales well as the number of simultaneous queries increases in the database.

2. its performance scales well as the data volume increases in the database.

3. its query speed up is especially better when the typical selectivity of the queries is low, i.e., queries only return small amount of information on average.

Query processings in MSDB are very fast because the signature databases are very small. Then the intermediate results are used to reduce search space in the main database to speed up query processing. The throughput of MSDB can also benefit from parallel processings in duplicated signature databases.

MSDBA can be implemented as an add-on software product to an existing relational database product because:

1. no parts of the main database need to be altered in order to implement MSDBA.

2. MSDBA technique is orthogonal to other query optimization and execution techniques so all other techniques can be used along with MSDBA.

Many aspects of the MSDB merit further research. For example, the grouping of data values, especially, both in text values as well as in numerical values, is critical to the performance of MSDB.

References

[1] Raghu Ramakrishnan. Database Management Systems. Beta Edition, 1996, ISBN 0-07-052522-6. McGraw-Hill Companies, Inc.

[2] P. Bosc and O. Pivert. Some approaches for relational databases flexible querying. *Journal of Intelligent Information Systems,* 1:323-354, 1992.

[3] Joseph M. Hellerstein. Predicate Migration: Optimizing Queries With Expensive Predicates. *Proc. of the 1993 ACM SIGMOD Int'l Conf. on Management of Data,* pp.267-276, 26-28 May 1993.

[4] E. Shekita and H. Young and K-L. Tan. Multi-Join Query Optimization for Symmetric Multi-Processors. *Proc. of the 19th Conf. on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Dublin,* August 1993.

[5] J-J. Yang and R. R. Korfhage. Query Optimization in Information Retrieval Using Genetic Algorithms. *Proc. of the 5th Int'l Conf. on Genetic Algorithms,* pp.603-613, Morgan Kaufman, July 1993.

[6] W. P. Yan and P-A. Larson. Performing group-by before join. *Proc. of the 10th IEEE Int'l Conf. on Data Engineering,* pp. 89-100, IEEE Computer Society Press, 1994.

[7] D. Kossmann, M. Franklin. A Study of Query Execution Strategies for Client-Server Database Systems. *Technical Report, CS-TR-3512 and UMIACS-TR-95-85,* University of Maryland, College Park, August, 1995.

[8] Ming-Ling Lo and Chinya V. Ravishankar. Towards Eliminating Random I/O in Hash Joins. *Proc. of IEEE 12th Int'l Conf. on Data Engineering.* Feb. 26-March 1, 1996, New Orleans, Louisiana.

[9] D. Duchamp. Issues in Wireless Computing. *Proc. of the Third Workshop on Workstation Operating Systems, IEEE,* Key Biscayne FL., April 1992, pp. 1-7.

[10] Dik Lun Lee, Chun-wu Leng. Partitioned Signature File: Design Issues and Performance Evaluation. *ACM Transactions on Information Systems, Vol. 7, No.2, April 1989, pp. 158 - 180.*

[11] P. ValDuriez. Join Indices. *ACM Transactions on Database Systems, Vol 12, No.2, pp. 218-246,*, June 1987.

[12] J. A. Blakeley, and N. L. Martin. Join Index, Materialized View, and Hybrid-Hash Join: A Performance Analysis. *Proc of 6th Int'l Conf. on Data Engineering, IEEE, pp. 256-263,* Feb 5-9, 1990, Los Angeles, CA.

[13] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data,* Montreal, Canada, 1996.

[14] J. Han, Y. Cai and N. Cercone. Concept-Based Data Classification in Relational Databases. *Workshop Notes of 1991 AAAI Workshop on Knowledge Discovery in Databases (KDD'91),* Anaheim, CA, July 1991, pp.77-94.

[15] Y. Cai, N. Cercone and J. Han. Attribute-Oriented Induction in Relational Databases, in *G. Piatersky-Shapiro and W. J. Frawley, pp. 213-228. AAAI/MIT Press.*

[16] J. Han, Y. Cai, and N. Cercone. Knowledge Discovery in Databases: An Attribute-Oriented Approach. *Proc. of 1992 Int'l Conf. on Very Large Databases (VLDB'92),* Vancouver, Canada, August 1992, pp. 547-559.

[17] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. on Knowledge and Data Engineering,* 5:29-40. 1993.

[18] J. Han, Y. Fu and R.T. Ng. Cooperative Query Answering Using Multiple Layered Databases. *Proc. of the 2nd Int'l Conf. on Cooperative Information Systems,* Toronto, Canada, May 1994. pp. 47-58.

[19] Blasgen, M. W., and Eswaran, K. P. Storage and access in relational databases. *IBM Systems Journal. Vol 16, No. 4,* 1977.

[20] H. F. Korth, A. Silberschatz. Database System Concepts. 1986, McGraw-Hill Computer Science Series. McGraw-Hill, Inc.

[21] K. Smith, M. Winslett. Entity Modeling in the MLS Relational Model. *Proc. of the 18th VLDB Conf., pp. 199-210*, Vancouver, British Columbia, 1992.

[22] X. Qian. A model-theoretic semantics of the multilevel relational model. *Advances in Database Technology - EDBT'94, Lecture Notes in Computer Science 779, pp. 201-214*, Jarke, M., Bubenko, J. and Jeffery, K. (editors), Springer-Verlag, 1994.

[23] F. Chen, R. S. Sandhu. The Multilevel Relational (MLR) Data Model. *Technical Report, ISSE-TR-95-101*, 1995, George Mason University.

[24] S. B. Yao. Approximating block accesses in database organizations. *Commun. ACM 20, 4(Apr. 1977), 260-261.*

[25] Z. Li, K. A. Ross. Fast Joins Using Join Indices. *Technical Report, CUCS-032-96*, 1996, Columbia University.