In this paper we present a multi-key index model that enables us to search a record with more than one attribute values in distributed database systems. Indices provide fast and efficient access of data and so become a major aspect in centralized database systems. Most of the centralized database systems use B+ tree or other types of index structures such as bit vector, graph structure, grid file etc. But in distributed database systems no index model is found in the literature. Therefore efficient access is a major problem in distributed databases. Our proposed index model avoids the query-flooding problem of existing system and thus optimizes network bandwidth.

**Keywords**: Relational database; distributed database; query optimization; multi-key index.

1. Introduction

A *distributed database* is a collection of data, which belongs logically to the same system but spread over the sites of a computer network. Every site of the network has a *Database Management Component* (DB) that is autonomous and capable of performing local applications. Every site also has a *Distributed Database Component* (DDB) that can execute global application that requires accessing data from several sites using a communication subsystem\(^1\)\(^2\). For global application the DDB acts like a *mediator*\(^3\) of distributed information retrieval systems. When a query is submitted, a mediator accepts
the query and transforms it into sub-queries and a composition query. It sends each sub-query to different sites or to wrappers of the sites based on homogeneity or heterogeneity of the sites. The sites execute the sub-query and send back the results to mediator. The mediator coordinates the arrival of the answers with the composition query. The composition query combines the answers into the final answer.

To support query execution in distributed information retrieval systems, different types of techniques are found in the literature such as, Search Enhanced by Topic Segmentation (SETS), Distributed Information Search Component (DISCO), broker implementation with grid files and partition hashing, Glossary-Of-Servers Server (GIOSS), hierarchical data index etc. but no single key or multi-key index model is found for distributed database system. In this paper we offer a multi-key based index model for distributed database system and provide comprehensive evaluations that demonstrate the viability of our index model.

The rest of the paper is organized as follows. Section 2 reviews some background knowledge. Section 3 gives problem overview, Section 4 presents our proposed multi-key index model, Section 5 shows performance comparison and Section 6 draws a conclusion.

2. Background Knowledge

Bitmap, graph structure, grid file etc. are the examples of multi-key indices implemented in database systems. A **bitmap index** on attribute A of a relation r consists of one bitmap for each value that A can take. Each bitmap has as many bits as the number of records in the relation. The i\(^{th}\) bit of the bitmap for value v\(_j\) is set to 1 if the record number i has the value v\(_j\) for attribute A. All other bits of the bitmap are set to 0.

In **graph structure**, a single node that contains a list for every attribute of the index represents each main file record. For every indexed attribute a node has two pointers. One points to the next node that has the same attribute value and the other points back to the index entry for the particular attribute value.

Nievergelt et al. describe a secondary key accessing technique, using grid. The grid file has a single **grid array**, and one **linear scale** for every search-key attribute. Search keys are mapped to cells. Each cell in the grid array has a pointer to the bucket that contains the search-key values and pointers to records. Multiple cells of the array can point to the same bucket.

3. Problem Overview

When a user submits a query, the source site sends the query message to all the participating sites of the system. In reality only few sites or even a single site (for point query) may produce non-empty results. Sending query to all the sites wastes bandwidth. Our objective is to send a query to minimum number of sites. Fig. 1 shows records of a car database at two different sites. To find the record Manufacturer = Honda, Model = Tempo and Color = Green the query is sent to site1 but not to site2.
4. Proposed Multi-Key Index Model

Our proposed index model, called Global Index (GI), is created and maintained by distributed database component (DDB) of distributed database management systems (DDBMS). There are two major challenges in creating and maintaining the proposed index model. Firstly, to keep succinct description, we call it record centroid, of each distinct combination of indexed attribute values of the global schema. Secondly, dynamic adaptation of the index i.e. insertion, deletion and update should be easily maintainable.

4.1. Record Centroid

The major design aim in creating a record centroid is to minimize the space requirement. The centroid is as like as a node of graph structure but it consists of one pointer for each indexed attribute values and one bit vector for site information. Fig. 2 shows one such centroid and Fig. 3 shows two such centroids from the car database example.

4.2. Global Index Creation

For easy maintenance we exploit the technique of grid file. In our model record centroids are kept in buckets and an $n$-dimensional grid array pointers point the buckets, $n$ is the number of indexed attributes. In our example, the global index is created on three attributes of manufacturer, model and color.
lexicographically, we might have two sets for manufacturer: Manufacturer $< G$, $G \leq$ Manufacturer; three sets for Model: Model $< K$, $K \leq$ Model $< R$, $R \leq$ Model; and two sets for color: Color $< H$, $H \leq$ Color. Thus the manufacturer Ford falls into the first partition (Ford $< G$). Similarly color Black falls into the first partition (Black $< G$) and model Pinto falls into the second partition ($K \leq$ Pinto $< R$). The partition points are held in linear scales. The set of three linear scales, one for each attribute, defines a grid on the three-dimensional attribute space. Fig. 4 shows grid array pointers pointing to buckets and Fig. 5 show one such bucket with its record centroids.

Fig. 4. Grid array pointing to buckets

4.3. Searching from Global Index

To search a record, the partitions are found from the linear scales. For example the attribute values of fourth record of site 1 fall into the partition 1, 2 and 1 respectively. So the corresponding record centroid falls into Bucket3 pointed by grid array index 2. In general if the grid array is $[i, j, k]$ and there are $P$, $Q$ and $R$ subsets in linear scales then the grid array index will be $(i - 1) * Q * R + (j - 1) * R + (k - 1)$.

Finding the appropriate bucket it is searched linearly for the record centroid. From Fig. 5, we see that the centroid of the record Manufacturer = Ford, Model = Mustang and Color = Black is found at second position of the bucket. The bit vector of the centroid is ‘10’ which means the record exists only at site1 but not at site2. Hence, the query is sent to site1 but not to site2. For range queries we follow the steps same to grid file 9.

4.4. Dynamic Adaptation

Whenever a record is inserted or updated to a site it is checked whether the combination of indexed attribute values already exists or not. If it is a new combination for that site, there are two possibilities. This combination may exist in other sites or this is entirely a new combination for all the sites. For the first case a record centroid exists in the global index and the site vector is updated. For the second case a new record centroid is created.
and stored in the global index of all the sites. While storing the centroid the bucket could be full of centroids. In such case bucket should be split into two buckets and centroids will be distributed between those two buckets. The splitting process is similar to the process of grid file\textsuperscript{9,10}.

When a record is deleted from a local database it is checked that whether there is any more such records in that database. If there is no such record then the site vector is updated to zero in the global index of all the sites. If all the bits are zero, the centroid is deleted from the bucket. To maintain reasonable storage utilization, two candidate buckets might be merged if their combined numbers of centroids fall below some threshold\textsuperscript{9}.

5. Performance Evaluation

Our measure of quality of multi-key index is query-processing cost directly tied with bandwidth cost. Also the storage space dedicated to the indexing metadata should remain within reasonable limits. We compare bandwidth i.e. amount of network traffic\textsuperscript{11} in bytes after the global index implementation to the present situation. For a query since the result set produced by the sites after index (AI) implementation will be same as before index (BI) implementation, we have not considered the bandwidth consumed by the result sets. We calculate the bandwidth requirements by following equations.

\[
BW_{AI} = \sum_{t} a(Q)n + \sum_{t} s(N-1),
\]

\[
BW_{BI} = \sum_{t} [a(Q)(N-1) + d(N-n-1)].
\]

Where the symbols have the following meaning:

- \(a(Q)\): Size of a query message
- \(n\): Number of sites that produce non-empty result set
- \(N\): Total number of sites
- \(d\): Size of empty result set
- \(s\): Average size of record centroids
- \(L(Q)\): TCP/IP header size + \(L(Q)\)

Fig. 6. Bandwidth comparison before and after index implementation

![Fig. 6. Bandwidth comparison before and after index implementation](image-url)
We assumed in average 30% sites produce non-empty result sets. Fig. 6 shows we gain more bandwidth for a system where query generation rate (70%) is higher than insertion, update and deletion. The gain is less when query generation rate is 50%. This is true because more and more bandwidth is consumed for index maintenance.

6. Conclusion

Traditional multi-key index implementation generally uses any one of the techniques such as bit vector, graph structure, grid-file etc. But in global index structure we exploit all of them. If there are \( p, q \) and \( r \) different values for three indexed key attributes then we have to keep information at most \( p \times q \times r \) number of records. Originally, in Graph Structure two pointers are used for each indexed attribute, but in our GI model only one pointer is used. Again the bit vector of site addresses replaces the data file pointer in the original graph node. So the total amount of space required for a centroid is almost half than the previous one. Since the centroids are stored in GI file in a fashion of Grid File we achieve the goal of searching a record in two disc file accesses. The main benefit of our proposed global index model actually lies with the fact that it reduces the number of sites that are to be consulted for global queries in distributed database system.

7. References