Dynamic Adaptation of Multi-key Index for Distributed Database System

M. S. HOSAIN†, M. A. H. NEWTON‡, M. M. RAHMAN‡‡
†Department of Computer Science & Engineering
The University of Asia Pacific
Road # 3/A, Dhanmondi, Dhaka – 1209
BANGLADESH
‡CSE Dept., Bangladesh University of Engineering & Technology, Dhaka – 1000
‡‡CSE Dept., Manarat International University, Dhaka – 1212
http://www.uap-bd.edu/cse/faculty/shazzad

Abstract: - Complex accessing structures like indices are a major aspect of centralized database for efficient record searching. We have proposed a multi-key index model, which enables efficient query execution in distributed database management system. In this paper we present algorithms that make our index model more adaptive in distributed environment because adaptability in a dynamic environment, space utilization, and operation speed are important criteria in assessing a multi-key file structure.

Key - Words: - Distributed database, Multi-key index, Dynamic adaptation and Grid file.

1 Introduction

A distributed database is a collection of data distributed over different computers in a network. Every site of the network has autonomous processing capability and can perform local applications. Every site also participates in the execution of global application, which requires accessing data from several sites using a communication subsystem [1, 2, 3]. For global application the query is transformed into sub-queries and the sub-queries are sent to different sites. The sites process the sub-queries independently, send back the result to the source node and the results are merged to produce the final answer [2, 4].

Different types of distributed index models [5, 6] are found in the literature to enhance query execution in distributed information retrieval systems. In [7] the authors have proposed an index model that is stored and distributed across the nodes of the network. It does not aim at answering complex database-like queries, but rather at providing practical techniques for searching data in a distributed has table (DHT). INS/Twine [8] is similar to this work that locates services and devices in large-scale environments using intentional (i.e., based on what one is looking for, not where it is located [9]) descriptions. Other techniques like Search Enhanced by Topic Segmentation (SETS) [10], Distributed Information Search Component (DISCO) [11], broker implementation with grid files and partition hashing [12], Glossary-Of-Servers Server (GlOSS) [13] etc are used to retrieve data from distributed information systems. We have proposed multi-key index model, which enhances query execution in distributed database system [14]. The index has two components *global index* and *local index*. The proposed global index is a multi-key based index model and is stored in all the sites. In this paper we present the techniques and algorithms, which enable the global index to be synchronized after insertion deletion and update of records in databases.

The rest of the paper is organized as follows. Section 2 presents background knowledge on multi-key index for distributed database system, section 3 gives dynamic adaptation of the index and finally section 4 draws a conclusion.

2 Background Knowledge

In distributed database a global query is sent to all the sites if the system can't locate the appropriate site according to *fragmentation schema* [2] and *allocation schema* [2]. This wastes bandwidth as well as takes longer time to process the query. Our proposed global index aims to minimize the number of sites that are to be consulted to process user queries and consequently to minimize the network traffic. To illustrate the index model briefly we give the same example given

in [14]. In our example we take two different sites of a car database that is shown in Fig.1.

Records at site 1

No	Manuf.	Model	Color	License
1	Ford	Pinto	Green	23023234
2	VW	Civic	White	23424223
3	BMW	Bug	Red	43543556
4	Ford	Mustang	Black	64354454
5	BMW	Mustang	White	45645634
6	Honda	Tempo	Green	23432567
7	VW	Civic	White	54654623
8	BMW	Bug	Red	34543545
9	Ford	Pinto	Green	54654634
10	Honda	Tempo	Green	54654632

Records at site 2

No	Manuf.	Model	Color	License
1	Ford	Mustang	White	23432421
2	VW	Civic	White	34655487
3	Ford	Pinto	Green	45654656
4	BMW	Pinto	Green	65765323
5	Ford	Mustang	White	32984934
6	Ford	Pinto	Green	56765712
7	Honda	Tempo	Red	54366554
8	VW	Civic	White	54765434
9	BMW	Pinto	Green	45645665
10	Honda	Tempo	Red	45654634

Fig. 1. Car database at two different sites

2.1. Record Centroid

The global index keeps succinct descriptions for all the distinct records of distributed database according to the indexed attributes. We call it record centroid [14]. The centroid has one bit vector for site information and one pointer for each indexed attribute. Fig. 2 shows one such centroid from the car database example.

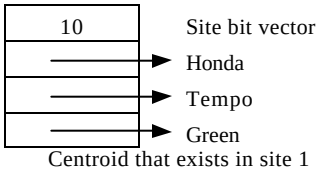

Fig. 2. Centroids with site addresses

2.2. Global Index Creation

The record centroids are kept in buckets and *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. If we divide the attribute values lexicographically, we might have two sets for manufacturer: Manufacturer < *G*, *G* <= Manufacturer; three sets for Model: Model < *K*, *K* <= Model < *R*, *R* <= Model; and two sets for color: Color < *H*, *H* <= 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* <= Pinto < *R*). The partition points are held in linear scales [3, 15]. The set of three linear scales, one for each attribute, defines a grid on the three-dimensional attribute space. Fig. 3 shows grid array pointers pointing to buckets and Fig. 4 show one such bucket with its record centroids.

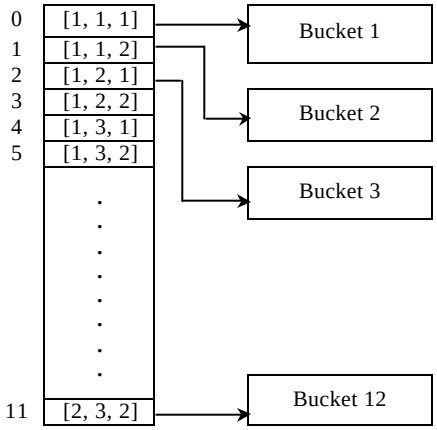

Fig. 3. Grid array pointing to buckets

3 Dynamic Adaptation

Every time a record is inserted, updated or deleted its centroid is updated in the global index if necessary. One of the major challenges of our index model is that it should be efficient and adaptable to highly dynamic environment [15], i.e., when there is a high rate of insertions and deletions. The adaptability in a dynamic environment, space utilization and operation speed are important criteria in assessing a multi-key index structure.

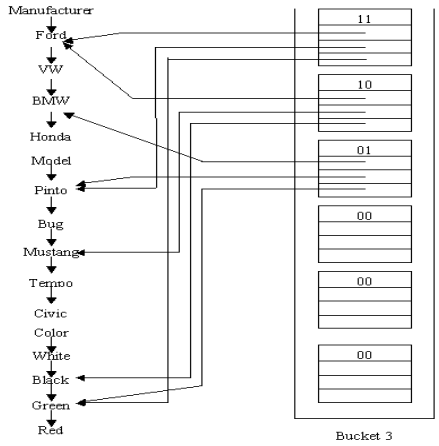

Fig. 4. Bucket 3

3.1 Global Index Insertion

Let a record with Manufacturer = Ford, Model = Mustang, and Color = Black is inserted into local database at site 2. If there already exists such record in site 2 and/or other sites then there is a centroid in global index. But if it is a new combination of values of indexed attributes there is no centroid in the global index. Site2 will send the values of this record to all other sites. Every site will now locate the appropriate bucket to store the corresponding centroid of the record. As mentioned above this record falls into grid array [1, 2, 1] and from this the centroid is stored in Bucket 3. Before saving the centroid the bit value for site2 in the site vector is made 1, i.e., the bit vector is 01.

While inserting a centroid, the bucket could be found full. In that case the bucket is split into two buckets and centroids will be distributed between those two buckets. The splitting process depends on two cases.

1. When only one pointer points to a bucket
2. When more than one pointer points to a bucket

To clarify the process we assume that, the Grid Array cells from [1, 1, 1] to [2, 3, 2] point one bucket each as shown in Fig. 3.

3.1.1 Case 1

According to linear scales the record with Manufacturer = BMW, Model = Pinto, and Color = Black falls into the bucket pointed by the grid array [1, 2, 1] i.e. the 3rd bucket and Fig. 3 shows the bucket is pointed only by one pointer. In that case one of the sub-ranges represented by the bucket contents must be divided. The splitting policies should consider the dimension (the axis, along which the grid block should be split) and the location (the point at which the linear scale is partitioned). The simplest splitting policies choose the dimension according to a fixed schedule. Other splitting policies may favor some attributes by splitting the corresponding dimensions more often than others [15]. We choose arbitrarily, the Manufacturer dimension. The location of a split on a linear scale need not necessarily be chosen at the midpoint of the interval. Little is changed, if the split point is chosen from a set of values, that are convenient for a given application; for example, months or weeks on a time axis [15]. In our example we choose a location at C. The corresponding linear scale is now Manufacturer (C, G), i.e. Manufacturer < C, C <= Manufacturer < G, G <= Manufacturer.

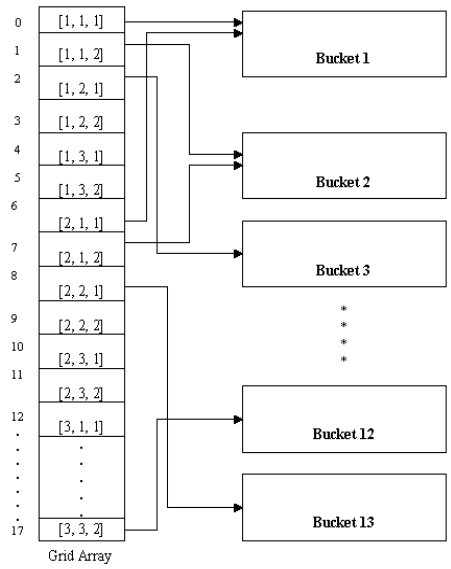

Fig. 5. Insertion of centroids when one pointer points to a bucket

Previously Bucket 1 was pointed only by one pointer: grid-array [1, 1, 1] but now pointed by two pointers: grid-array [1, 1, 1] and grid-array [2, 1, 1], because the records that fall into group Manufacturer < G, Model < K, Color < H now fall into two groups. The groups are Manufacturer < C, Model < K, Color < H and C <= Manufacturer < G, Model < K, Color < H. Similarly both grid-array [1, 1, 2] and grid-array [2, 1, 2] point to Bucket 2 and so on. Now we allocate a new bucket, Bucket 13, and distribute the

records of Bucket 3 to Bucket 3 and Bucket 13 according to the following two groups:

1. Manufacturer < C, K <= Model < R, Color < H
2. C <= Manufacturer < G, K <= Model < R, Color < H

Group1 is put in Bucket 3 and group2 in Bucket 13. So grid-array [1, 2, 1] points to Bucket 3 and grid-array [2, 2, 1] points to Bucket 13. Fig. 5 shows the allocation of Bucket 13 and the rearrangement of pointers in grid-array.

3.1.2 Case 2

Now insert a record that falls into Bucket 2 that is pointed by two pointers: grid-array [1, 1, 2] and grid-array [2, 1, 2]. If the bucket is full then the linear scales are not split rather a new bucket, Bucket 14, is allocated and centroids are distributed according to the following groups:

1. Manufacturer < C, Model < K, H <= Color
2. C <= Manufacturer < G, Model < K, H <= Color

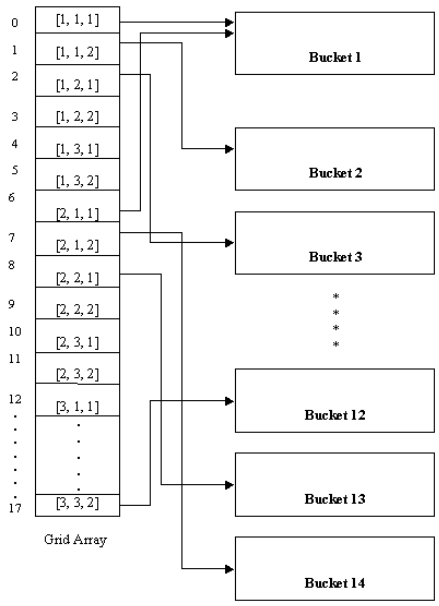

Fig. 6. Insertion of centroids when more than one pointer point to a bucket

If group1 is kept in Bucket 2 and group2 in Bucket 14 then grid-array [1, 1, 2] points to Bucket 2 and grid-array [2, 1, 2] points to newly created bucket. Fig. 6 shows the allocation of Bucket 14 and rearrangement of pointers in grid-array. Simulation results show that as son as the number of inserted centroids reaches a small multiple of the bucket capacity, the average bucket occupancy is around 70 percent for both the growing file and the steady state file.

3.1.3 Insertion Algorithm

The algorithm to insert a centroid into the global index is given below:

```
Insert GI ( inCentroid ){
  Search GI ( inCentroid )
  If inCentroid exists in global index Then
    turn on the bit of the corresponding site
    return
  Else
    If the bucket is no full Then
      insert the inCentroid into the bucket
    Else If bucket is full Then
      find the number of pointers point the bucket
      If the bucket is pointed by only one pointer Then
        randomly select any one of the linear scales
        divide it into its middle
        add a new bucket into the index file
        distribute the centroids according to new linear scales
        make necessary changes of the grid array pointers
        put the inCentroid in the corresponding bucket
      End If
      Else If the number of pointers are more than one Then
        add a new bucket into the index file
        make necessary change in grid array pointers
        distribute centroids according to grid array pointers
        put the inCentroid in the corresponding bucket
      End If
    End If
  End If
}
```

3.2 Global Index Deletion

When a record is deleted from a local database its corresponding bit in the site vector is made zero and the result is published to other sites. If all the bits in the site vector are zero then the centroid is deleted from the bucket. To maintain reasonable storage utilization, two candidate buckets might be merged if their combined number of records falls below some threshold. The records would be moved into one of the buckets and pointers to the other reassigned to it. The empty bucket would be removed from the file. There are two kinds of merging: bucket merging and merging of cross sections in the grid directory. It is

needless to reduce the directory size as soon as possible, because it will soon grow back to its earlier size.

3.2.1 Deletion Algorithm

The algorithm to delete a centroid is given below:

```
Delete GI ( outCentroid ){
   Search GI ( outRec )
   If the outRec is found in the bucket Then
      Turn off the bit of the corresponding site
      If all site bits are zero Then
         delete the centroid from bucket
         If the no of centroids fall below some threshold Then
            move centroids into one bucket
            reassign the bucket pointers
         End If
      End If
   End If
}
```

4 Simulation Results

To analyze the behavior and performance of the global index we have been done a simulation. The performance of an index file is determined by two criteria: processing time and memory utilization. In view of the fact, that the grid file holds the two-disk-access principle and the grid directory only needs a small part of the space, we only look at the average bucket occupancy of the global index. The average bucket occupancy does not need to be close to 100 percent as well as it should not be arbitrary small. For the growing file, which results from repeated insertions, the simulation result is given in Fig. 7.

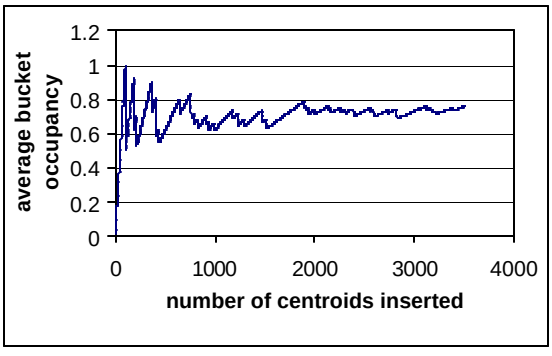

Fig. 7. Average bucket occupancy for a growing file

In this simulation the bucket capacity is taken as 100. From Fig. 7 it is found that as soon as the number of inserted centroids reaches a small multiple of the bucket capacity, the average bucket occupancy shows a steady state behavior with small fluctuations around 70 percent. Similarly, for the steady state file, where in a long run the number of insertions is equal to the number of deletions, the average bucket occupancy will show steady state behavior with small differences for different percentage of merging-threshold policy.

5 Conclusion

The multi-key index for distributed database system is designed to handle efficiently a collection of centroids with a modest number of search attributes. Within this usage environment it combines quite a few of the better properties of multi-key file structures, such as, high data storage utilizations of 70 percent; smooth adaptation to the stored contents; a directory, which is quite compact and efficient space utilization.

References:

[1] D. Bell and J. Grimson, *Distributed Database Systems*, Addison-Wesley, Wokingham, 1992.

[2] Stefano Ceri and Ginseppe Pelagatti, *Distributed databases Principals & Systems*, McGrawHill Book Company, 1984.

[3] Abraham Silberschatz, Henry F. Korth and S. Sudarshan, *Database System Concepts*, 3rd ed., The McGraw Hill Companies, 1997.

[4] Anthony Tomasic, Louiqa Raschid and Patrick Valduriez, Scaling access to heterogeneous database with DISCO, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, No. 5, 1998.

[5] Maria Wahid Chowdhury, Chowdhury Mofizur Rahman and Md. Humayun Kabir, Distributed Index: Algorithms and Models, *Proc. of the 4th International Conference on Computer and Information Technology*, Dhaka, Bangladesh, December 2000.

[6] Jinyang Li, Boon Thau Loo, Joseph Hellerstein, ,Frans Kaashoek, david Karger and Robert Morris, On the Feasibility of Peer-to-Peer Web Indexing and Search, *Proc. of 2nd International Workshop on Peer-to-Peer Systems*, Berkely, CA, February 2003.

[7] P. A. Felber, E. W. Biersack, L. Garces-Erice, K.W. Ross and G. Urvoy-Keller, Data Indexing and Querying in P2P DHT Networks, *ICDCS*, Tokyo, Japan, 2004.

[8] M. Balazinska, H. Balakrishnan and D. Karger, INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery, *Proc. of the 1st International Conference on Pervasive Computing*, August 2002.

[9] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan and Jeremy Lilley, The Design and Implementation of an Intentional Naming System, *Symposium on Operating Systems Principles*, 1999.

[10] Mayank Bawa, Roberto J. Bayardo Jr. and Rakesh Agrawal, SETS: Search Enhanced by Topic-Segmentation, *Proc. of the 26th Annual ACM Conference on Research and Development in Information Retrieval*, Berkeley, California, August 2003.

[11] Anthony Tomasic, Louiqa Raschid and Patrick Valduriez, Scaling access to heterogeneous database with DISCO, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, No. 5, 1998.

[12] Anthony Tomasic, Luis Gravano, Calvin Lue, Peter Schwarz and Laura Haas, Data structures for efficient broker implementation, *ACM Transactions on Information Systems*, Vol. 15, No. 3, July 1997.

[13] L. Gravano, H. Garica-Molina and A. Tomasic, Gloss: Text-source discovery over the Internet, *ACM Transactions of Database Systems*, Vol. 24, No. 2, 1999, pp. 229- 264.

[14] Md. Shazzad Hosain and Muhammad Abdul Hakim Newton, Multi-Key Index for Distributed Database System, *International Journal of Software Engineering and Knowledge Engineering*, Vol. 15, No. 2, May 2005, pp. 433 – 438.

[15] Nievergelt, J. Hinterberger and H. Sevcik, The Grid File: An Adaptable Symmetric Multi-Key File Structure, *ACM Transactions on Database Systems*, Vol. 9, No. 1, 1984, pp. 38-71.