Migration Policies of a Main Memory Index Structure for Moving Objects Databases

Kyounghwan An and Kwangsoo Kim
Telematics Research Division,
Electronics and Telecommunications Research Institute,
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, Republic of Korea
{khan, enoch}@etri.re.kr

Abstract: To manage and query moving objects efficiently in MMDBMS, a memory index structure should be used. The most popular index structure for storing trajectories of moving objects is 3DR-tree. The 3DR-tree also can be used for MMDBMS. However, the volume of data can exceed the capacity of physical memory since moving objects report their locations continuously. To accommodate new location reports, old trajectories should be migrated to disk or purged from memory. This paper focuses on migration policies of a main memory index structure. Migration policies consist of two steps: (i) node selection, (ii) node placement. The first step (node selection) selects nodes that should be migrated to disk. The criteria of selection are the performance of insertion or query. The second step (node placement) determines the order of nodes written to disk. This step can be thought as dynamic declustering policies.

Keywords: LBS, Moving Object, Main Memory Database.

1. Introduction

Recently, the need for LBS (Location Based Services) is increasing due to the widespread of mobile computing devices (e.g. PDA, cellular phone, and notebook computer) and positioning technologies (e.g. GPS and RFID). There are many applications that need to manage and query a large number of "moving objects" in LBS. Since the moving objects should report their locations frequently, a database system should be able to handle a huge number of updates and queries efficiently. In such an environment, traditional disk-resident DBMSs cannot be used since the number of updates and queries exceeds the capacity of the disk. To overcome the limitation, main memory resident DBMSs (henceforth MMDBMS) should be used to manage moving objects.

In MMDBMS, to manage and query moving objects efficiently, a memory index structure should be used. There were several index structures for moving objects databases [4, 5]. The most popular index structure for storing trajectories of moving objects is 3DR-tree. However, if the index structure is used in MMDBMSs, the volume of data can exceed the capacity of physical memory since moving objects report their locations continuously. To accommodate new location reports, old trajectories should be migrated to disk or purged from memory.

This paper focuses on migration policies of a main memory index structure. Migration policies consist of two steps: (i) node selection, (ii) node placement. The first step (node selection) selects nodes that should be migrated to disk. The criteria of selection are the performance of insertion or query. The second step (node placement) determines the order of nodes written to disk. This step can be thought as dynamic declustering policies. By using suggested migration policies, we can obtain following advantages. First, a main memory index structure can store location reports continuously without stopping the system. Second, the performance of insertions and queries can be improved.

The remainder of this paper is organized as follows. In section 2, we examine related works. In section 3, we define the problems when migrating nodes to disk and section 4 presents migration policies that consist of two steps. A summary of the paper is presented in section 5.

2. Related Works

There were studies on storing data separately in different media [1] and migrating data [2, 3] according to the volume of data, the frequencies of use, and the restrictions on processing data.

The study [1], which stores data in different media, stores hot data in main memory based DBMS while cold data in disk based DBMS. For example, bank applications may store account information in main memory while customer information in disk. However, it can not handle the situation if data accumulate over time and exceeds physical memory capacity.

The study [2], which migrates nodes, stores current data (nodes) in disk while migrates past data (nodes) into optical media. Time-Split B-tree in [2] migrates a node into optical media when a node splits in time axis. Since it migrates a node at a time it cannot be applied to our environment which requires minimum disk I/O. Also, it is inefficient in main memory environment since there exist repeated data in the nodes.

The study [3], which migrates data, migrate data into different media according to the semantic of data by administrator or DBMS. However, it does not suggest the method migrating index nodes.

3. Problem Definition

Migration policy migrates part of main memory index from memory to disk when the volume of index structure exceeds given limitation. After the migration, nodes exist both in main memory and in disk. In this section, we define the problems when migrating nodes into disk.

1) Pointers in nodes

Since nodes can exist both in main memory and in disk after migration, a node can have two physical pointer types: (i) main memory pointer, (ii) disk pointer. Logical types of pointers can be classified into four categories according to the direction they point.

Table 1. Logical Types of Pointers

Direction	Name of Pointer
Memory → Memory	m2m pointer
Memory → Disk	m2d pointer
Disk → Memory	d2m pointer
Disk → Disk	d2d pointer

When a memory resident node (henceforth memory node) points to other memory node, it is called m2m pointer. Usually main memory based indices have m2m pointers. When a disk resident node (henceforth disk node) points to other disk node, it is called d2d pointer. Usually disk based indices have d2d pointers. When a node points a node of other type, it is called m2d or d2m pointers. Two pointers are new types that do not exist in the previous indices. In those pointers, d2m pointers have the following problems.

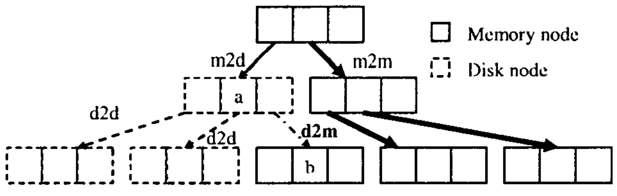

Fig. 1. Problem of d2m pointer

First, after parts of the nodes migrate to disk by migration policies (in Fig 1, nodes of dotted line), several types of pointers can exist. In Fig 1, there is a d2m pointer. The d2m pointer has the following problems. First, if node b should be migrated to disk, node a should be loaded into memory to update a pointer. Second, if node b should be referenced by a query, node a should be loaded before accessing node b.

It is inefficient to load a disk node to access a memory node. To prevent the situation, migration policy should consider avoiding d2m pointers.

2) Migration

A migration policy should minimize disk I/O when migrating nodes into disk or referencing disk nodes by queries. Also it should operate non-blocking mode since location reports should be processed in a limited time constraint. To achieve the goals the followings are should be considered.

- Migration Time

Main purpose of migration policies is to ensure memory space when location reports are inserted into an index structure. If free memory space is below the given threshold, migration of nodes from memory to disk should occur. Migration time depends on both the threshold and granularity of migration.

- Node Selection

Selecting a candidate node for migration is important since it affects the query performance. If queries access disk nodes frequently, the performance will deteriorate. It would be better to choose nodes that will not be referenced by queries. In general, more recent trajectories will be referenced frequently than old trajectories. Node selection algorithm should consider the properties of moving objects databases.

- Granularity of Migration

If migration policy writes a node at a time, the system will suffer from disk I/O. Thus, it is necessary to write several nodes at a time. If granularity is large, more memory space will be needed since the memory space that is occupied by migration nodes can not be used for insertion algorithm of index structure. However, large granularity will be more efficient when loading nodes into memory since some dynamic clustering will be applied while migrating nodes.

4. Migration Policies

1) Steps of Migration Policies

The following figure shows the steps of the migration policies.

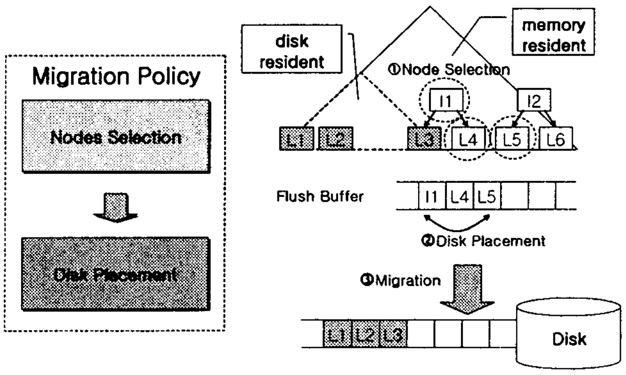

Fig. 2. Migration Policy

When free memory space is below given threshold, migration occurs. The first step of migration policy is to select candidate nodes to migrate. If nodes are selected, the second step is to rearrange nodes in flush buffer to improve query performance. Finally, rearranged nodes are written to disk. Before the final step, pointers are translated from memory pointers to disk pointers. All steps operate in a non-blocking mode by flush daemon. Node selection policy and disk placement policy can be tightly coupled or loosely coupled. It means nodes can be selected considering disk placement or not.

2) Node Selection Policy

There can be several methods to select candidate nodes. To avoid d2m pointer, node selection policies should not select a node until all child nodes become disk resident. Figure 3 shows the example.

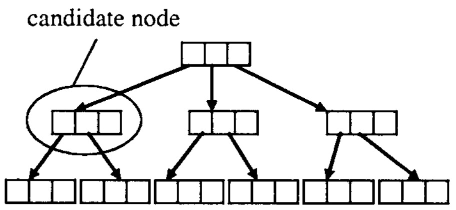

Fig. 3. Principle of Node Selection

In fig 3, although a non-leaf node is selected as a candidate node, it cannot be migrated to disk since it has child nodes that reside in memory. If the candidate node migrates to disk, d2m pointers will be generated. Node selection policy should proceed until it finds a node that does not make d2m pointers.

In this paper, we suggest three node selection policies that follow above principle.

▪ **NSP1**: Selection by Creation Time
The first node selection policy is to choose a node according to the creation time of a node. Since it is simplest approach, there is no overhead when selecting nodes.

▪ **NSP2**: Selection by Node Time
The second node selection policy is to choose a node according to the time value of node's MBB. Since the probability of referencing a node is not by the creation time of a node but by the time value of MBB, it is more appropriate to take this approach. To select a candidate node, it traverses index structure until it finds a node which has oldest time value of MBB.

▪ **NSP3**: Selection by LRU
The final approach is to choose a node according to LRU policy. In this approach, a least recently used node is selected. This approach can be good if insertions and queries are made on main memory. However, it can be inefficient if insertions and queries are made on disk since it may not be good for the disk placement policy.

3) Node Placement Policy

There can be several methods to place candidate nodes into disk. The purpose of node placement policy is to reduce disk I/O when queries are made on disk resident nodes.

In this paper, we suggest two node placement policies.

▪ **NPP1**: Ordering by Node Selection
The first approach is to order nodes by the order of node selection. This approach can be used if there is low probability of selecting disk resident nodes. This approach can be very inefficient if there are many chances to reference disk resident nodes (old trajectories).

▪ **NPP2**: Ordering by Index Structure
This approach is to order nodes according to index structure. This approach is good when there are many chances to reference disk resident nodes. Since it is possible to load several nodes at a time (bulk loading), disk I/O time is reduced than other approach. In this approach, the granularity becomes subtree. The following figure shows the example.

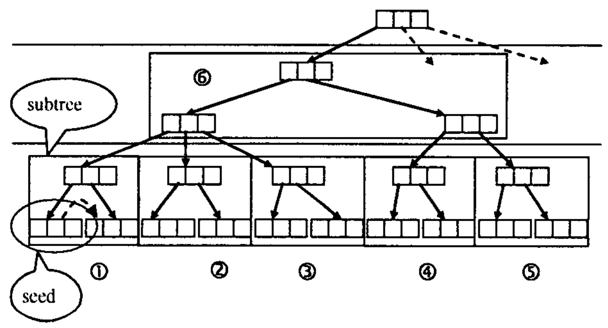

Fig. 4. Ordering by index structure

In fig 4, granularity of migration is decided as two level of the tree (deciding granularity is explained later). If a seed node is selected, a subtree is decided containing the seed node. After the subtree is decided, nodes are ordered from parent node to child nodes according to the order of breadth first search.

4) Migration Time

Migration occurs when free memory space becomes under given threshold. Following formula explains migration time.

$$\sum_{n=0}^{h-1}(F \times U)^n \geq \frac{M}{S_p} - N \times \alpha \qquad (\alpha \geq 1) \qquad (1)$$

In formula (1), let h be the tree height, F be fanout, and U be node utilization. M means free memory space, S_p means a size of one page, N is the number of page that is needed to migrate (granularity). α (stability factor) means the number of free space for migration. The more α becomes greater, the more the system will operate stable.

5) Granularity of Migration

Granularity of Migration (N) is determined to prevent the system from blocking. To guarantee stable operation of the server, the number of moving objects and the frequency of reporting locations should be considered. The following equations give us the granularity of migration.

Let m be the number of moving objects to be managed and T_i be the interval of reporting locations. Then λ is reporting rate and shown in eq. (2).

$$\lambda = \sum_{i=1}^{m} \frac{1}{T_i} \tag{2}$$

Let F be the fanout of the tree and U be the utilization of the tree. N is the number of page needed to migrate and we want to obtain its value. The rate of accessing disk (W) is expressed in eq. (3).

$$W = \frac{\lambda}{F \times U} \times \frac{1}{N} \tag{3}$$

Let T_{seek} be the time for seeking right position in disk and $T_{transfer}$ be the time for transferring one page. Then N (granularity) should be determined by the following formula.

$$C = W \times (T_{seek} + N \times T_{transfer}) \leq 1 \tag{4}$$

6) Structure of Nodes

After the migration, an index structure can have several types of pointers. To distinguish pointers, we use bitmap indicating the type of a pointer in the head of a node. When search algorithm traverses the tree, it uses bitmap information.

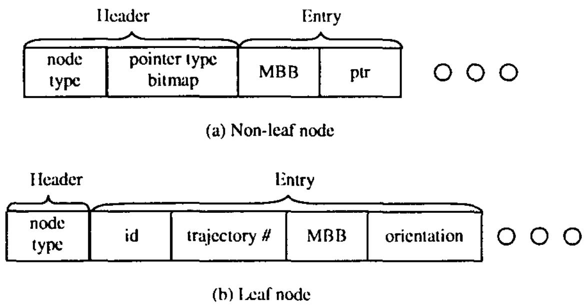

Fig. 5. Structure of a Node

In case of the leaf node, the structure is similar to that of STR-tree[4]. It has ID of line segment in a trajectory, three dimensional MBB (Minimum Bounding Box), and orientation of the line segment.

In case of the non-leaf node, pointer type bitmap is included. Pointer type bitmap indicates the types of pointers in the entries.

In case of both non-leaf node and leaf node, node type field is included in the header. It indicates whether the node exist in memory or both memory and disk. A node exists both in memory and disk when it is loaded from disk by a query. Migration occurs when a node exist only in memory.

5. Conclusions

Moving objects reports their locations continuously over time. Since the volume of trajectories can be large to be stored in main memory, part of index structure should be migrated to disk. In this paper, we suggested migration policies that consist of two steps: (i) node selection, (ii) disk placement.

By using our algorithm, we can achieve the following things. First, main memory based moving objects database can operate continuously without blocking the system. Second, when insertions and queries are made on main memory and disk, it can process it efficiently since node selection and disk placement considers the performance. Our further work is to implement the system and compare the performance of suggested algorithms.

References

[1] D.Gawlick and D.Kinkade, "Varieties of concurrency control in IMS/VS Fast Path," Data Eng. Bull., vol.8.no.2,pp.3-10, June 1985

[2] D. Lomet and B. Salzberg, "Access Methods for Multiversion Data", Proc. of SIGMOD, 1989

[3] M.Stonebraker, "Managing persistent objects in a multi-level store," in Proc. ACM SIGMOD Conf., Denver, CO, May 1991, pp2-11

[4] D.Pfoser, C.S.Jensen, and Y.Theodoridis, "Novel Approaches in Query Processing for Moving Objects,"In Proc.of the VLDB Conference, pp.395-406, 2000

[5] N.Beckmann, H.-P.Kriegel, R.Schneider and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," Proceedings of ACM SIGMOD Int'l. Conf. on Management of Data, pp. 322-331, 1990

[6] Güting, R., Böhlen, M., Erwig, M., Jensen, C. S.,Lorentzos,N.,Schneider, M., and Vazirgiannis, M.: A Foundation for Representing and Querying Moving Objects. ACM Transactions on Database Systems, to appear, 2000

[7] Hector Garcia-Molina and Kenneth Salem. Main memory database systems: An overview. IEEE Trans. Knowledge Data Eng., 4:509-516, 1992

[8] Li Chen,Rupesh Choubey and Elke A.Rundensteiner, "Merging R-Trees: Efficient Strategies for Local Bulk Insertion", Geoinformatica Volume6 Issue1, 2002

[9] T.J.Lehman and M.J.Carey,"A Study of index structures for main memory database management systems," in Proc.12th Conf. on VLDB,Aug.1986