International Journal of Networked and Distributed Computing, Vol. 3, No. 2 (April 2015), 89-98

Parallel Indexing Scheme for Data Intensive Applications

Kenta Funaki, Teruhisa Hochin, Hiroki Nomiya
Department of Information Science, Kyoto Institute of Technology, Goshokaido-cho, Matsugasaki
Sakyo-ku, Kyoto, 606-8585, Japan
E-mail: m3622036@edu.kit.ac.jp, {hochin, nomiya}@kit.ac.jp

Hideya Nakanishi, Mamoru Kojima
National Institute for Fusion Science, Oroshi-cho
Toki, Gifu, 509-5292, Japan
E-mail: nakanisi@nifs.ac.jp

Abstract

This paper proposes a parallel indexing scheme of a large amount of data in order to resolve the issues about time limitation. Three kinds of computing-nodes are introduced. These are reception-nodes, representative-nodes, and normal-nodes. A reception-node receives data for insertion. A representative-node receives queries. Normal-nodes retrieve data from indexes. Here, three kinds of indexes are introduced. These are a whole-index, a partial-index, and a reception-index. In a partial-index, data are stored. In a whole-index, partial-indexes are stored as its data. In a reception-index, additional data are stored. The reception-index is moved to a normal-node, and becomes a partial-index. The proposed scheme is also a data distribution scheme for shortening the insertion time. A reception-node accepts additional data even if the index is already built.

Keywords: Multi-dimensional index, Parallel processing, Insertion performance, Distributed index

1. Introduction

In recent years, information diversification grows the importance of multimedia databases. As a multimedia database contains a lot of data, its fast retrieval is required. Feature values of multimedia data are usually used in the fast retrieval. As they are represented with multi-dimensional data, the multi-dimensional index is inevitable for the fast retrieval of multimedia data. The R-tree family is a popular index structure for the multi-dimensional data.[1, 2, 3] In the R-tree, an object is represented with a minimum bounding rectangle (MBR)

Published by Atlantis Press
Copyright: the authors

in a multi-dimensional space. MBRs of objects are also represented with an MBR. This recursive nature constitutes a tree structure. By using this tree structure, search space can efficiently be narrowed.

In addition, advances in measurement technology and those of equipment make it possible to get big data in a variety of fields easily. Therefore, a multi-dimensional index having good performance in inserting and retrieving data is required. For instance, in the fusion experiment of National Institute for Fusion Science,[4] a large amount of measurement data are generated every few minutes. This means that data must be inserted within the limited time constraints.

The methods of implementing a large multi-dimensional index by parallelization have been proposed.[5] The ends of the proposed methods are to insert data within the limited time constraints, to retrieve data quickly, and to store a large amount of multi-dimensional data. It has experimentally been revealed that data can be inserted within the limited time by file splitting. It has also been shown that parallel processing to indexes could accelerate the retrieval performance. Parallel indexing could also resolve the problem of the limitation of the file size.

Although it has been shown that parallel processing of indexes could accelerate the retrieval performance, the acceleration is limited under the configuration where a storage is shared by many cores. Because many cores give different requests to a disk storage, the overhead of the movement of the magnetic head could not be ignored.

This paper proposes a parallel indexing scheme that is good for retrieval performance as well as insertion one. In this scheme, data are sequentially inserted into an index on a computing-node called a reception-node. This index is called a reception-index. When a reception-index becomes full, it is moved to another computing-node for improving retrieval performance by parallel processing. A reception-index is treated as an entry in a multi-dimensional index on the computing-node. A query is parallelly processed by computing-nodes, each of which manages its own disk storage. As a reception-node concentrates on the insertion of data, and creates a reception-index in its disk storage, the insertion performance may become good. As computing-nodes parallelly search candidate data from their own disk storages, the retrieval performance may also become good.

Remaining of this paper is as follows. Section 2 describes related works. Section 3 proposes a parallel indexing scheme. Section 4 gives some considerations. Finally, Section 5 concludes the paper.

2. Related Works

Many spatial index structures for implementing multi-dimensional indexes have been proposed. The R-tree[2] is one of the most popular methods. The R*-tree[3] is an improvement to the R-tree. Improving the split algorithm makes a tree efficient.

2.1. *Parallel R-tree*

Many parallel spatial indexing methods based on the R-tree have been proposed. The elementary idea of the parallel R-tree is dividing data to multiple disks.[1] Whole data are divided into several data and stored to each disk. In each disk, independent R-trees are created using allocated data. On the other hand, the method of dividing whole R-tree structure into several nodes and storing them to each disk is called disk stripping. In the R-tree with supernodes,[1] every node of an R-tree is divided into several subnodes, and subnodes are stored into several disks. In the Multiplexed R-tree (MX R-tree),[1] an R-tree is divided into subtrees. A subtree is stored into a disk. Several dividing schemes have been proposed.

Performance of the parallel R-tree can be accelerated using multiple processors in addition to multiple disks. In the leaf-based declustering scheme[1]

Copyright: the authors

and Master-Client Rtrees,[5] the master-server holds non-leaf nodes, while leaf nodes are held by slave-servers. The Global Parallel index R-tree (GPR-tree)[6] uses a global index tree shared by multi-computers. In the method of the GPR-tree, each processor manages its own subtree. It holds a copy of the other subtrees of the whole tree, and exchanges the messages to maintain the consistency. The Replicated Parallel Packed R-tree (RPP-Rtree)[7] replicates the whole R-tree across all the computers. In this method, leaf nodes and inner nodes are stored into each computer, but the actual data are stored separately. The Distributed Parallel R-tree (DPR-tree)[8] partitions the whole data into partition sub-region, and allocates the partition sub-region among each computer. Each computer creates an independent R-tree structure using allocated data.

From the point of view of the logical tree structure, there are two approaches. One uses independent R-trees.[1, 8] The other uses one large logical R-tree.[1, 5, 6, 7] When independent R-trees could make search space be narrowed, the retrieval performance will be improved. In the large logical R-tree, all of accesses begin with the computer holding the root node of the R-tree. This will degrade the retrieval performance. From the point of view of holding data, there are also two approaches. One holds a copy of an R-tree. The other does not hold it. Holding a copy improves the retrieval performance. The insertion performance, however, degrades in general.

Faloutsos et al. have proposed and evaluated the following node distribution methods.[9]

(i) *Round Robin*: This method distributes data to multiple disk in rotation.

(ii) *Minimum Area*: In this method, each disk has same and minimum area by data distribution.

(iii) *Minimum Intersection*: This method minimizes the overlap of nodes in the same disk.

(iv) *Proximity Index*: In this method, each disk calculates the proximity between the own MBR and the MBR of the new data value. The new data value is allocated to high proximity disk. Proximity of two rectangles is measured by the proportion of queries that retrieve both rectangles and calculated by the following equation:

$$proximity = \frac{number\ of\ queries\ retrieving\ both}{total\ number\ of\ queries} \quad (1)$$

In practice, it is not calculated by the above definition, but calculated by the following method. First, each rectangle is divided into each dimension. In the first dimension, whole section is normalized to [0-1]. If two rectangles overlap with each other, let "δ" be the length of overlap. The proximity of ith dimension is calculated by the following equation:

$$proximity_i = \frac{1}{3} \times (1 + 2 \times \delta) \quad (2)$$

On the other hand, if two rectangles do not overlap with each other, let "Δ" be the distance between them. The proximity of ith dimension is calculated by the following equation:

$$proximity_i = \frac{1}{3} \times (1 - \Delta)^2 \quad (3)$$

The proximity of each dimension is determined in the same manner and the proximity of the two rectangles is calculated by multiplying all proximity of the dimension:

$$proximity = \prod_i proximity_i \quad (4)$$

Faloutsos et al. showed that the Proximity Index has high retrieval performance by experimentation.[9]

Copyright: the authors

2.2. *Expansion of the index by file split*

In general, when an object is inserted into an R*-tree index, the node, which the object is inserted into, is searched, and the MBRs of nodes on the path from the root node may be modified. When the node exceeds the capacity of the number of objects, the node is split. If the number of objects stored in a tree is huge, the numbers of nodes required to be modified, to be integrated, and to be divided become large. Therefore, the processing time of insertion and retrieval will increase. When there are constraints on the insertion time, a number of data inserted into the index is limited. This constraint can often be required in data intensive applications, where massive data are required to be stored in a few seconds.[10]

In order to build a large-scale multi-dimensional index, the two methods by file splitting have been proposed.[10] One is the method that an index is created one by one. When the size of an index reaches the limitation, a new index is created. The other is the method that several indexes are created in advance. Data are inserted into the indexes according to the Round-robin scheme. Fig. 1 shows the outline of these methods. Sequential creation has the advantage that there is no need to decide the number of indexes and that it can accept more data than estimated in advance. However, the data in each index are biased when the similar data are continuously inserted. In contrast, Round-robin distribution can avoid bias in typical data set. However, the number of indexes must be decided in advance.

In Ref. 10, the performance of the two methods was evaluated. When inserting data to the index, it is necessary to read the index structure in order to determine the insertion point. Round-robin distribution

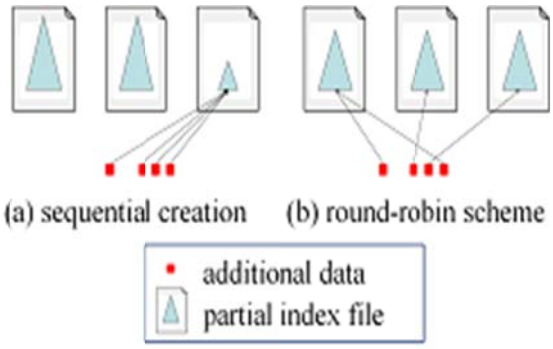

Fig. 1. File splitting methods.

reads all indexes for insertion but sequential creation reads only one index to determine the insertion point. In addition, if the main memory has released the index data, insert operation needs long time in order to read the index data from hard disk drive (HDD). Therefore, sequential creation is faster than Round-robin scheme in insertion. In retrieval, these methods can use multiple threads. However, parallel retrieval needs a lot of time than serial retrieval. Parallel retrieval reads multiple files and accesses the multiple HDD fields. Thus, parallel retrieval requires a back-and-forth motion of the magnetic head of HDD and a long time. Moreover, file division shortens retrieval time without multi-thread if the main memory released the index data. Accordingly, reading time from HDD affects the retrieval time strongly and the time of disk seek affects the reading time from HDD.

3. Proposed Method

We try to address to the issue described above by dividing the index.

Copyright: the authors

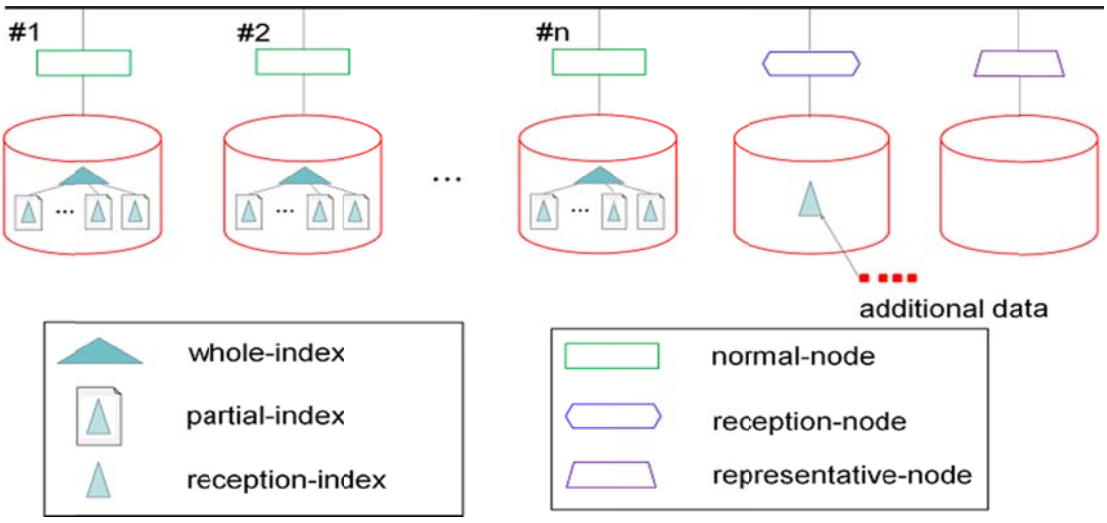

Fig. 2. Method overview.

3.1. *Overview*

Fig. 2 shows the outline of the proposed method. It is assumed that the system has several computing-nodes. Each computing-node has its own disk storage. Computing nodes are connected to one another through bus.

A computing-node, which is called a reception-node, has a role of receiving data newly inserted. This node inserts the data received into an index, which is called a reception-index. Other computing-nodes store data in the form of indexes. A computing node has a whole-index and partial-indexes. In a whole-index, a partial-index is managed as a rectangle, which is represented with an MBR, and is an entry of a node of an R-tree-family index. This computing-node is called a normal-node.

The insertion algorithm is shown in Fig. 3. When additional data are inserted to the system, the additional data are inserted into a reception-index using the insertion algorithm of R-tree (Line 1). When the number of the data stored into the reception-index reaches the number determined in advance (Line 2), data insertion is interrupted. The MBR covering a whole of the reception-index is compared with the MBRs of the whole-indexes. After the suitable whole-index is decided (Line 3), the reception-index on the reception-node is moved to the normal-node that has the suitable whole-index (Line 4). The normal-node that receives the reception-index rebuilds the whole-index. The reception-index becomes a partial-index on the normal-node. After that or in parallel, the reception-node, which may still be receiving new data, creates a new reception-index (Line 5), and inserts them into the newly-created reception-index. The algorithm of decision of suitable normal-node is shown in Fig. 4. The whole-index, which is the most suitable for the reception-index RI, is searched from the whole-indexes of all of the normal-nodes. The normal-node having the whole-index obtained is returned to the caller. The meaning of suitability depends on the distribution schemes described later. The algorithm of moving reception-index is shown in Fig. 5. The reception-index RI is moved to the normal-node NN from the reception-node having RI at Line 2. The MBR of RI is obtained at Line 3. This MBR is inserted into the whole-index at Line 4.

Copyright: the authors

```
Algorithm insert_data(data D, reception-index RI)

/* insert a data item D into the reception-node that has
     reception-index RI*/

1.  insert(D, RI)
2.  IF the number of the data stored into RI reaches the
     number determined in advance
3.  NN = find_normal-node(RI)
4.  move_index(RI, NN)
5.  RI = new R-tree index
6.  ENDIF
```

Fig. 3. The insertion algorithm at a reception-node.

```
Algorithm move_index(reception-index RI, normal-
          node NN)

/* move RI to NN */

1.  partial-index = RI
2.  NN.store(partial-index)
3.  mbr = get_mbr(RI)
4.  NN.whole-index.insert(mbr)
```

Fig. 5. The algorithm of moving reception-index.

A computing-node, which is called a representative-node, receives queries. When a query is posed to the

```
Algorithm find_normal-node(reception-index RI)

/* search the normal-node that has suitable whole-
     index WI for RI */

1.  FOR each WI
2.      IF WI suits for RI than candidate-index
3.          candidate-index = WI
4.      ENDIF
5.  ENDFOR
6.  RETURN candidate-node having candidate-index
```

Fig. 4. The algorithm of finding a normal node.

representative-node, all of the normal-nodes and the reception-node search data suitable to the retrieval condition specified in the query. The data retrieved are sent to the representative-node from the normal-nodes and the reception-node. This node puts the data together, and returns them to a user. Fig. 6 and Fig. 7 show the algorithms of retrieval at a representative-node and a normal-node, respectively. The query Q is forwarded to all of the normal-nodes and a representative-node as shown in Fig. 6. At a normal-node, partial-indexes, which may have candidate data, are identified by using a whole-index. The candidate data are obtained from the partial-indexes identified.

Up to here, a reception-node and a representative-node are assumed to exist apart from the normal-nodes. A computing-node may serve both as a normal-node and a reception-node or representative-node. A reception-node could also be a representative-node.

More than one reception-node can be introduced. When two reception nodes exist, a reception-node can

Copyright: the authors

receive data soon after the reception-index is built up by the other one, which can move it to a normal-node. Similarly, more than one representative-node can be introduced. In this case, multiple queries can be processed in parallel.

3.2. *Distribution Scheme*

In the proposed scheme, data are always inserted into the reception-index that exists apart from the normal-nodes. Therefore, there is no need to read the normal-indexes. Insertion time is expected to be reduced. In addition, reception-indexes are moved to the normal-nodes in a distributed manner. This distribution would reduce the number of hits per computing-node and the retrieval time.

Two distribution schemes adopting the following criteria are proposed.

(i) *Maximum area*: Each computing-node calculates an expanded MBR area for receiving the reception-index. The reception-index is allocated to the normal-node whose MBR's area expansion is the maximum one. Fig. 8 shows three examples of area expansion. In Fig. 8, each chain-lined rectangle shows a node (normal-node), each white solid rectangle shows the MBR of a whole-index, each black solid rectangle shows the MBR of a reception-index, and each dashed rectangle shows area expansion, respectively. In these examples, the case (a) has the largest area expansion of them and the reception-index will be moved to the node shown in Fig. 8(a).

(ii) *Proximity*: Each computing-node calculates a proximity between the own MBR of whole-index and the MBR of the whole reception-index. The reception-index is allocated to the computing-node whose MBR of the whole-index has low proximity

```
Algorithm search(query Q)

/* representative-node sends Q to normal-nodes and
     reception-node */



1.  answer = {}

2.  FOR each normal-node NN

3.        answer += search_normal(Q, NN)

4.  ENDFOR

5.  answer += retrieve(Q, reception-node)

6.  RETURN answer
```

Fig. 6. The algorithm of search at a representative-node.

```
Algorithm search_normal(query Q, normal-node NN)

/* search in NN */



1.  answer = {}

2.  candidate = retrieve(Q, whole-index)

3.  FOR each partial-index PI

4.        IF candidate includes get_mbr(PI)

5.                answer += retrieve(Q, candidate)

6.        ENDIF

7.  ENDFOR

8.  RETURN answer
```

Fig. 7. The algorithm of search at a normal-node.

Copyright: the authors

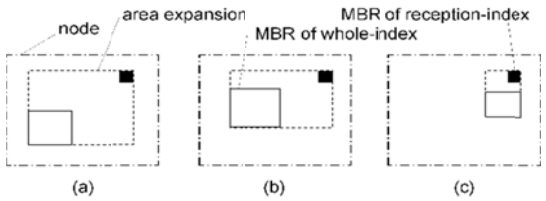

Fig. 8. Examples of area expansion.

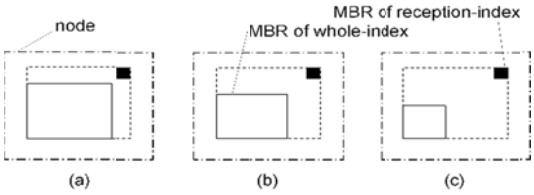

Fig. 9. Examples of proximity.

with the whole reception-index. Fig. 9 shows the three examples of proximity. The notations shown in Fig. 9 are the same as those shown in Fig. 8. In each example, a whole-index and a reception-index are disjoint. Therefore, the farther the two rectangles in each dimension are, the lower the proximity is. The reception-index is moved to the node shown in Fig. 9(c), because two rectangles of the case (c) have the longest distance in these examples.

4. Consideration

In the proposed scheme, data are inserted into a single file, which is for an index, rather than multiple files. The overhead of the insertion becomes minimum because the overhead of reading data can be minimized. This is based on the research result obtained by Funaki et al.[10]

The proposed scheme moves a set of data items stored in an index to one of the processing nodes for improving the retrieval performance. As a processing node manages its own disk storage, there is no overhead caused by multiple requests posed by multiple nodes. This configuration could improve the retrieval performance.

A reception-index is moved to a normal-node. In a whole-index in a normal-node, a reception-index is treated as a single data item rather than a set of data items. Only one insertion is required in a whole-index. If a set of data items was inserted into a whole-index, n insertions are required, where n is the number of the data items. In this case, much time is considered to be taken in the insertion. The proposed scheme avoids this degradation of the insertion.

When a reception-index is inserted into a whole-index, the reception-index is placed as far as possible in the whole-one. This will avoid the concentration of data to a normal-node. The data stored in partial-indexes are placed in a distributed manner over normal-nodes. As many normal-nodes will join the query processing, it could be highly paralleled. This will also improve the retrieval performance.

In the parallel query processing, the collection of candidates sent from normal-nodes is often the bottle neck of the processing. The candidates are usually sorted. Much time and much space are required. In such a case, it is recommended that more than one reception-node is introduced. A reception-node could concentrate on the processing of its own query.

This scheme is similar to the Multiplexed R-tree (MX R-tree).[9] In Ref. 9, retrieval time is evaluated. Insertion time is, however, not considered. Additional insertion is neither considered. On the other hand, in the proposed scheme, additional data can be accepted by a reception-node even if all indexes have been created once.

From the viewpoint of parallel processing, the best situation is that every normal-node has the same number of partial-indexes. This situation is, however, not guaranteed. Some normal-nodes have a lot of data, while others have a few data. The worst case is that a normal-node has N - n partial-indexes, and each of the

Copyright: the authors

others has only one partial-index, where there are totally N partial-indexes on n+1 normal-nodes. This will degrade the retrieval performance because only one normal-node can process the query in most cases. The initial placement of the whole-indexes may affect the balance of data. Clarifying the method of keeping the balance of data is in future work.

5. Conclusions

Information diversification grows the importance of multimedia databases. As the fast retrieval of multimedia data is required, the multi-dimensional index for the fast retrieval using the feature values is required. For instance, in the fusion experiment of National Institute for Fusion Science, a large amount of measurement data are generated every few minutes. There is a major issue of the limitation of the insertion time. Measured data must be inserted into a database in a few minutes.

This paper proposed a parallel indexing scheme in order to resolve the issue described above. This scheme has three kinds of node; normal-node, reception-node, and representative-node. This scheme has three kinds of indexes; whole-index, partial-index, and reception-index. Each whole-index manages the partial-indexes that are stored into each normal-node. A partial-index manages the actual data. A reception-index manages the additional data until the additional data are moved to a normal-node. A reception-node accepts the additional data, and creates a reception-index. A representative-node works in the retrieval. As a reception-node concentrates on the insertion of data, and creates a reception-index in its disk storage, the insertion performance becomes good. As computing-nodes parallelly search candidate data from their own disk storages, the retrieval performance also becomes good.

Future work includes the implementation and evaluation of this scheme. In this paper, two distribution schemes are proposed. Comparing these schemes is also included in future work.

References

1. Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, *R-trees:Theory and Applications* (Springer, 2006), pp.151-159.
2. A. Guttman, R-trees: A dynamic index structure for spatial seatching, in *Proc. 1984 ACM SIGMOD International Conference on Management of Data,* (1984) pp. 47-57.
3. N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger, The R*-tree: an Efficient and Robust Access Method for Points and Rectangles, in *Proc. 1990 ACM SIGMOD Conference on Management of Data*, (1990) pp. 322-331.
4. National institute for fusion science. [Online]. Available: http://www.lhd.nifs.ac.jp/
5. B. Seeger and P. A. Larson, Master-Client R-trees - a New Parallel R-tree Architecture, in *Proc. 11th International Conference on Scientific and Statistical Database Management (SSDBM'99)*, (Cleveland, OH 1999) pp. 68-77.
6. X. Fu, D.Wang and W.Zheng, GPR-tree, a Global Parallel Index Structure for Multiattribute Declustering on Cluster of Workstations, in *Proc. International Conference on Advances in parallel and Distributed Computing (APDC'97)*, (1997) pp. 300-306.
7. Mutenda. L, and M. Kitsuregawa, Parallel R-Tree Spatial Join for a Shared-Nothing Architecture, in *Proc. the 1999 International Symposium on Database Applications in Non-Traditional Environments*, (1999) pp. 423-430.
8. Y. Zhou, Q. Zhu, and Q. Liu. DPR-Tree: A Distributed Parallel Spatial Index Structure for High Performance Spatial Databases, in *Proc. International Conference on Earth Observation Data Processing and Analysis (ICEODPA)*, (2008) pp. 72853A-72853A-7.
9. C. Faloutsos, and I. Kamel, Parallel R-trees, in *Proc. the 1992 ACM SIGMOD International Conference on Management of Data*, (1992) pp.195-204.
10. K. Funaki, T. Hochin, H. Nomiya, H. Nakanishi, and M. Kojima, Parallel indexing of large malti-dimentional data, in *Proc. 1st ACIS Inteructional Symposium on*

Copyright: the authors

Applied Computing and Information Technology (ACIT 2013), (2013) pp. 324-329.

Copyright: the authors