An Efficient Multidimensional Index Structure for Parallel Environments

Koung Soo Bok
Department of Computer Science
Korea Advanced Institute of Science and Technology, Daejeon, Korea

Seok Il Song
Department of Computer Engineering
Chungju National University, Chungju, Korea

Jae Soo Yoo*
Department of Computer and Communication Engineering
Chungbuk National University, Cheongju, Korea

ABSTRACT

Generally, multidimensional data such as image and spatial data require large amount of storage space. There is a limit to store and manage those large amounts of data in single workstation. If we manage the data on parallel computing environment which is being actively researched these days, we can get highly improved performance. In this paper, we propose a parallel multidimensional index structure that exploits the parallelism of the parallel computing environment. The proposed index structure is nP(processor)-n×mD(disk) architecture which is the hybrid type of nP-nD and 1P-nD. Its node structure in-creases fan-out and reduces the height of an index. Also, a range search algorithm that maximizes I/O parallelism is devised, and it is applied to k-nearest neighbor queries. Through various experiments, it is shown that the proposed method outperforms other parallel index structures.

Keywords: *Multidimensional Data, Parallel Computing, Range Search, k-Nearest Neighbor Search, Index Structure.*

1. INTRODUCTION

In the past couple of decades, multidimensional index structures play a key role in modern database applications such as GIS (Geographic Information System), LBS (Location Based Service), content based image retrieval system and so on. The applications commonly are required to manipulate multidimensional data. For example, GIS store and retrieve two-dimensional geographic data about various types of objects such as a building, a river, a city and so on. Also, MLS systems provide clients with the current locations of moving objects such as mobile phones. The locations of moving objects are represented as points in the two-dimensional space. To satisfy the requirements of the modern database applications, various multidimensional index structures have been proposed. There are space partitioning methods like Grid-file[1], K-D-B-tree[2] and Bang-file[3] that divide the data space along predefined or predetermined lines regardless of data distributions. On the other hand, such as R-tree[4], R+-tree[5], R*-tree[6], X-tree[7], SR-tree[8], TV-tree[9] and CIR-tree[10] are data partitioning index structures that divide the data space according to the distribution of data objects inserted or loaded into the tree. Besides, Hybrid-tree[11] is a hybrid approach of data partitioning and space partitioning methods, VA-file[12] uses flat file structure, and [13] uses hashing techniques.

As mentioned above, many researchers have studied multidimensional index structures to improve retrieval performance in various ways. However, there are bounds in improving retrieval performance with a single index structure. Also, for large amount data a single index structure may show insufficient retrieval performance. To solve these problems, several index methods using parallelism of processors or disk I/Os have been proposed[14, 15, 16, 17, 18, 19, 20]. These parallel multidimensional index structures can be classified into 1P-nD and nP-mD where P and D are processor and disk respectively. In 1P-nD architecture, multiple disks are connected to one processor so as to improve performance through parallel disk I/Os. However, there is only one channel between a processor and multiple disks so loading data to memory is processed in serial. On the other hand, in nP-mD architectures, multiple disks are connected to multiple processors. Therefore, the index structures using this architecture exploit parallelism of processors and disk I/Os.

In this paper, we propose a parallel multidimensional index structure that exploits the parallelism of the parallel computing environment. The proposed index structure is nP-n×mD

*Corresponding author. E-mail : yjs@chungbuk.ac.kr
Manuscript received Jan 17, 2005; accepted Feb 24, 2005

architecture which is the hybrid type of nP-nD and 1P-nD. That is, there are multiple processors and each processor have multiple disks. Our node structures increase fan-out and reduce the height of an index tree. Also, range search algorithm that maximizes I/O parallelism is presented. To our knowledge, existing parallel multidimensional index structures hardly consider k-NN(K-Nearest Neighbor) queries. We propose new k-NN search methods that are proper to our index structures. Through various experiments, it is shown that the proposed method outperforms other parallel index structures.

The rest of this paper is organized as follows. In section 2, we describe existing parallel index structures. In section 3, we present the detailed description of our parallel high-dimensional index structure. In section 4, the results of performance evaluation is presented. Finally, we conclude in section 5.

2. RELATED WORKS

Existing parallel multidimensional index structures are classified into 1P-nD and nP-mD types. In 1P-nD architecture, multiple disks are connected to 1 processor so as to improve performance through parallel disk I/Os. MXR-trees[14] and PML-trees[16] are 1P-nD parallel index structures. These improve the performance of multidimensional index structures using the parallelism of disk I/O. In [14], the requirements for improving range searches are presented. The one is *minLoad*. When the load of processing queries is light, searchers should access as few nodes as possible. Consequently, queries with small selectivity should activate as few disks as possible. The other is *uniSpread*. Nodes that accessed by a query should be distributed over the disks as uniformly as possible. Consequently, queries with large selectivity should activate as many disks as possible. Three approaches are proposed to distribute an R-tree over multiple disks. First approach construct d independent R-trees. Second approach stripes super-node which consists of d pages on the d disks by striping pages. The last approach is MXR-tree (MultipleXed R-tree). In this approach, a single R-tree is constructed. Each node is spanned one disk page. Nodes are distributed over the d disks, with pointers across disks.

PML-tree proposed in [16] uses native space indexing with a disjoint space decomposition method. The disjoint space decomposition method does not allow overlapping intermediate MBR(Minimum Bounding Region)s. The PML-tree eliminates the extra search paths of the R-tree and the leaf node redundancy of the R+-tree by distributing data objects into multiple data spaces. Two data distribution heuristics, which distribute data over the multiple disks evenly, are proposed and implemented. These index structures improve search performance with exploiting disk I/O parallelism. However, there is only one channel between a processor and disks so loading data to memory is processed in serial. In the nP-mD architecture, multiple disks are connected to multiple processors. nP-mD parallel index structures are constructed based on special environments such as NOW (Network of Workstation). Therefore, nP-mD index structures can use parallelism of processors and disk I/Os. MR-Tree, MCR-Tree and parallel R-tree based on DSVM (Distributed Shared Virtual Memory), GPR-Tree and Parallel VA-file are nP-mD parallel index structures.

MXR-tree proposed in [14] is shown in Fig. 1. One master server contains all internal nodes of the parallel R-tree. The leaf level at the master does not hold the leaf level of the global tree, but (MBR, site-id, page-id) tuples for each global leaf level node. The leaf nodes of the global tree are distributed across the other servers. The (site-id, page-id) is used to locate a page and server that contains the page. Once a query is sent to the master, the master searches the internal nodes of the MR-tree and produces a list of all (site-id, page-id) pairs needed. The constructed list and the query are sent to sites containing required pages. Each site then retrieves the page from disk and sends qualifying rectangles back to the master.

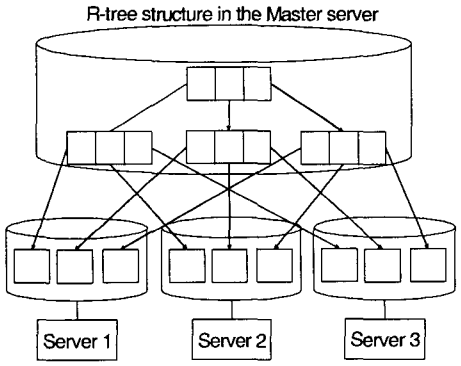


Fig. 1. Architecture of MR-tree

MCR-tree(Master-Client R-tree) proposed in [17] reduces communication messages of MR-tree. One master and multiple clients are connected through computer network. The data structure of the master server is almost same to that of MR-tree, i.e. only internal nodes are stored at the master server. Unlike the MR-tree, only the (MBR, site-id) pairs are needed at the leaf level of the master server. Each client builds a complete R-tree for the portion of the data assigned to it. In the MCR-tree, there is redundant information stored compared to the MR-tree. This redundant information reduces the overhead at the master and global communication costs.

A query is sent to the master and is processed locally as in the sequential case. When a leaf node is reached, the query MBR is sent to the client site designated in the leaf node. As soon as a client receives a request, it starts processing the query autonomously. All data are retrieved and returned back to the master. The master adds the client site id to a list that keeps track of which clients are working on the query. Since the clients work autonomously, a maximum of one request is sent to each client. The master continues searching until either all clients are notified of the query or no more MBRs intersecting the query are found. The master then waits for answers and collects the qualifying data items sent back by the clients.

3. THE PROPOSED INDEX STRUCTURE

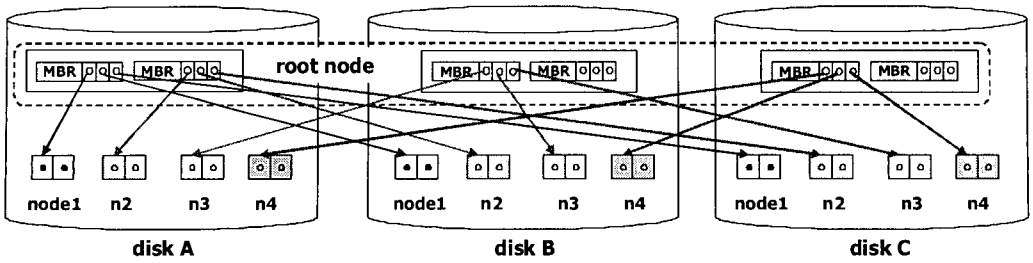


Fig. 3. Index structure of a disk group

Fig. 2 shows the architecture of the proposed parallel index structure. Disks are grouped into the number of servers evenly. The groups are assigned to servers. One primary server coordinates search process and others are normal servers that process index operations. R-trees are distributed to servers and each server including primary server has an independent R-tree. The R-tree of each server is distributed to multiple disks. We call this architecture as nP-n×mD type. We exploit the parallelism of CPUs and each CPU uses the parallelism of multiple disk I/Os.

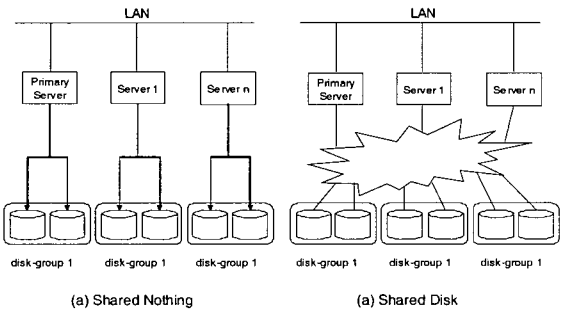


Fig. 2. Architecture of the proposed parallel index structure

Each server manages a disk group and the disk group contains an independent R-tree. Fig. 3 shows the R-tree of a disk group. As shown in the Fig., a node in the index structure is distributed to disks in the group, i.e., a node consists of the pages of disks. An entry in the node contains child node's MBR and the pointers of those pages that consist of the node. In the Fig., the first entry of the root node points the *node 1*. The *node 1*consists of the first pages of disk A, B and C, so the entry must have the pointers of these pages and the MBR of the *node 1*.

The benefits of our architecture are as follows. First, similar data are declustered across multiple disks in the group. Since the entries in a node are distributed to multiple disks, declustering effects are maximized. Second, the height of index tree is reduced. The size of a node is determined by the page size and the number of disks in the group. As the node size increases, it takes more time to load a node into memory. However, because the index structure can load the node in parallel, the loading
time is not a problem. In R-tree family, overlaps between nodes reduce the retrieval performance. The height of a tree is one of the factors to increase overlaps. As the height of a tree becomes higher, more overlaps may be caused. Finally, in multidimensional index structures, as the dimension increases the number of nodes to be accessed increases. That is, the number of node accesses is large when processing range search or k-NN search. Subsequently, in parallel multidimensional index structure, uniSpread is much more important than minLoad. The proposed index structure read all pages that consist of a node, so it maximizes the uniSpread.

3.1 Insert

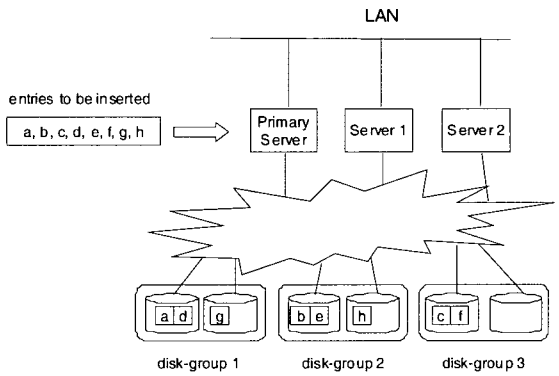


Fig. 4. Insertion of entries

Fig. 4 shows the process of entry insertion. Assume that we insert entries *a, b, c, d, e, f, g* and *h* sequentially. The entries are declustered across disk-groups in round robin fashion, i.e., *a*is inserted into *disk-group 1*, *b* is inserted into *disk-group 2* and so on. Various declustering techniques have been proposed, but in high-dimensional data sets, the performance gap among them is not so large. Also, round-robin technique is easy and cheap to implement. In that reason, we choose round-robin technique as the declustering method. Entries assigned to each group are inserted into the index structure of the group. In the first phase, we find a proper node to insert a new entry. When a node is located, we check whether the node has enough space to accommodate the entry. Then, if overflow occurs, we start split process.

When processing node split, we need to carefully allocate pages for newly created node. In general, nodes in multidimensional index structures are not always full. Consequently, we cannot fully obtain disk I/O parallelism when accessing index nodes. To relieve this problem, we place the pages of two nodes (old node and new node) in different disks as much as possible so as to increase disk I/O parallelism when processing range search. We will describe our range search algorithm in the next section. Fig. 5 shows node split process. In *node 2(n2)*, overflow occurs. To split *node 2*, we

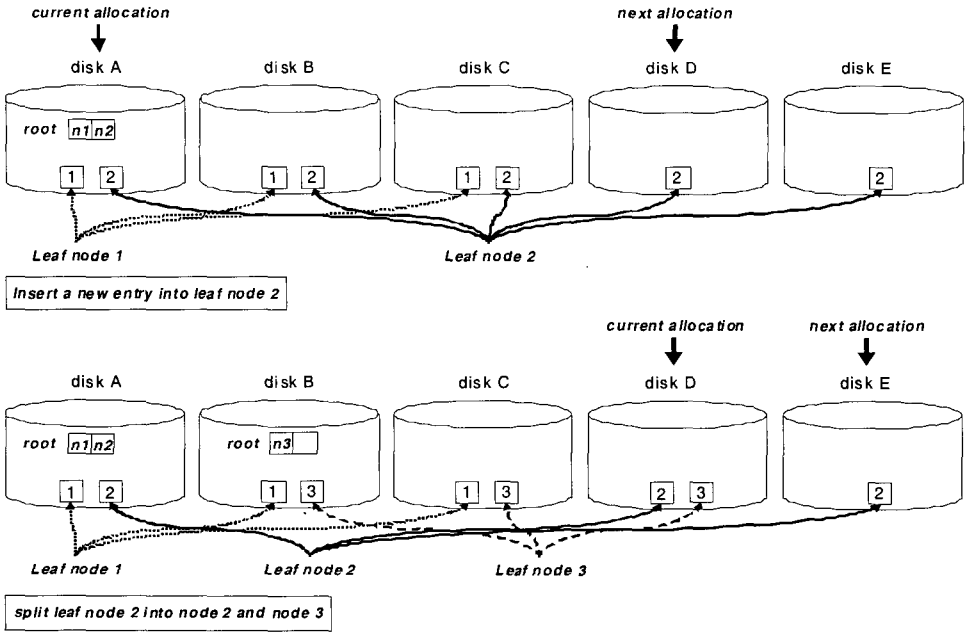


Fig. 5. Node split

assign a new node (*node 3*) and move partial entries of *node 2* to *node 3*. When allocating pages to *node 2* and *node 3*, we preferentially choose disks that have the smallest number of allocated pages. In the lower Fig., disk D and E have the smallest number of allocated pages, so pages for *node 2* and *node 3* are allocated from these two disks. First, we allocate three pages from D, E and A sequentially, and then allocate three pages from B, C, and D sequentially. Fig. 6 shows the pseudo code of our insert algorithm.

3.2 Search

3.2.1 Range Search

Range search algorithms for multidimensional index structures have been mentioned in several researches [1, 10]. Searchers take multiple paths when processing range search. That is, multiple nodes may be selected as next nodes to visit. Existing range search algorithms visit the selected child nodes sequentially. Fig. 6 shows the process of existing range search algorithms. A searcher chooses entries *2, 4* and *7* from root node that are overlapped with the searcher's predicate. The searcher visit child nodes that are pointed by *2, 4* and *7* sequentially. To read node 2, the searcher must access disk *A, D* and *E*
since the pages of node 2 are distributed disk *A, D* and *E*.

In the similar fashion the searcher visit node *4* and *7*. Total number of disk accesses is the sum of the number of disk accesses to read root node and leaf nodes. The number of disk accesses to read root node is *1* and that of leaf nodes is *3*. Therefore, the total number of disk accesses to process the range query is *4*.

Our new range search algorithms use different approaches to load child nodes. Once child nodes to visit are determined, we make a page loading plan according to which disks are involved to load child nodes. Fig. 6 describes how to make the page loading plan. In the Fig., *A3* means third page of disk *A*. There are 8 pages to be read. We cluster these pages into groups consists of pages from different disks. For example, pages *A3, B3, C3, D4* and *E1* in *GRP1* are from different disks. It means that those pages can be read at one I/O time. Also, *A5, D4* and *E2* in *GRP2* are from different disks, so we can read them in one I/O time. If we load pages in this way, only two disk I/Os are needed to load leaf nodes. One disk I/O is saved compared to the previously mentioned method.

3.2.2 k-NN Search

Existing parallel multidimensional index structures hardly consider k-NN search. However, k-NN queries are important in modern database applications. In this paper, we propose three k-NN algorithms and through experiments we show which one is the best.

3.2.2.1 Type 1

The primary server distributes a k-NN query to servers and each server processes the k-NN query independently. Then, the servers return the k results to the primary server.

The primary server filters the results from servers and makes final k results. The response time is the sum of the longest time among servers' response time and the time to filter servers' results. This method is simple and easy to implement. However, we may not use disk I/O parallelism like our range search algorithm because of the properties of the k-NN algorithm. When processing range search, a searcher chooses all child nodes to visit next that are overlapped with query predicates before going down to next level. Therefore, we can

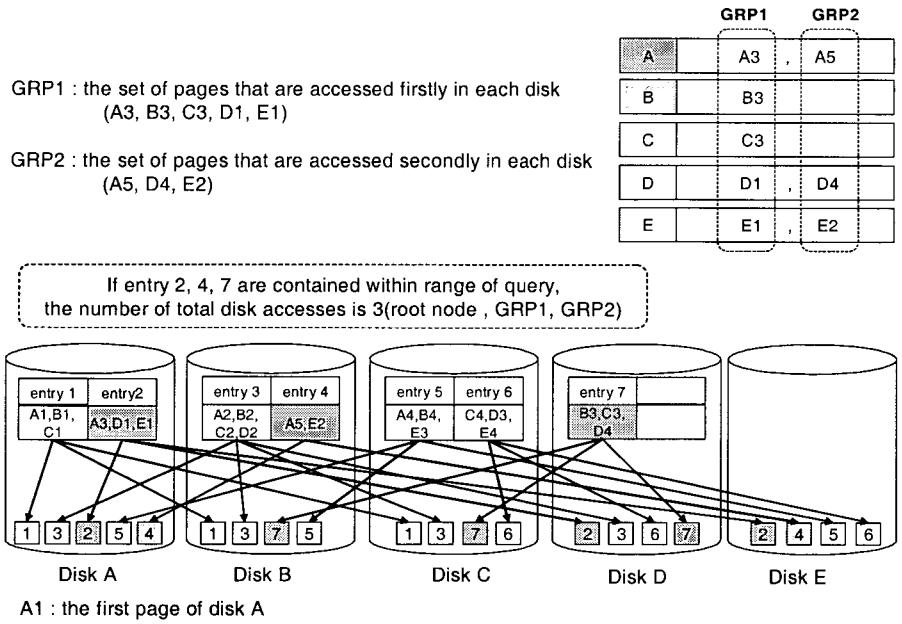


Fig. 6. Example of range search

make a page loading plan and save disk I/Os. However, in the existing k-NN search algorithm, all child nodes to visit next are not determined definitely but just one child node is determined. Consequently, we cannot make a page loading plan as in our range search algorithm.

3.2.2.2 Type 2

In the second method, the primary server transforms k-NN queries to range queries. Once a k-NN query is arrived from client, the primary server processes k-NN query partially. When the primary server gets first k results, it calculates the distance between k'th element and query point of the given k-NN query. It makes a range query with the distance. The range query is distributed to servers and the servers process the range query and return results. The primary server gathers the results from servers and makes k results. The time to process k-NN query partially is quite short. Since servers can process the transformed range query, this method can get parallelism of range search algorithm. However, the transformed range query may become larger and reduce the overall performance.

3.2.2.3 Type3

In the third method, once the primary server receives a k-NN query from clients, it sends the query to all servers. The servers execute partial k-NN queries with the received query, transform the k-NN query torange query similar to the primary server of type 2 and return the transformed range query to the primary server. Then, the primary server redistributed the transformed query to servers. The servers process the range query and return their results to the primary server. Finally, the primary server makes k results from server's results.

4. PERFORMANCE EVALUATION

The simulation platform is Sun Enterprise 250 with 1GB main memory and Solaris 2.7. Simulation programs are developed with gcc 2.8 compiler. Table 1 shows simulation parameters. N_{da} means that the total number of disk accesses to perform a query. Assume that the number of disk accesses to read pages in parallel from different disks is 1. The response time to process a range query and type 1 k-NN query is calculated by the equation, $max(RT_i)$ + *filtering time* + *total message size*× T_{comm}, where i = 0~ N_{server}, *filtering time* is the time to filter results from server and make final results and *total message size* is the size of total communication messages between the primary server and each server. The response time of type 2 and 3 k-NN queries is calculated by the following equation, *query transform time* + *response time of a range query*. The *query transform time* of type 2 is calculated by the equation, N_{da} *for a partial k-NN query*×T_{diskIO} + T_{cpu} *for a partial k-NN query*. The query transform time of type 3 is calculated by the equation, $max(RT_i$ *for partial k-NN query)* + *filtering time* + *total message size*× T_{comm}, where i= 0~ N_{server}. We assume the value of T_{comm} and T_{diskIO} as in Table1 according to [21].

We use uniformly distributed 100,000 data with 10 ~ 80 dimensions. We measure response time and total number of disk accesses of a query to compare the retrieval performance of our index structure with existing parallel multidimensional index structures. We perform several experiments in various environments. We present the results of experiments with varying dimension, the number of disks and page size. We compare our proposed index structure with MCR-tree. To our knowledge, the MCR-tree is the most recently proposed nP-mD parallel index structure and shows best performance

among existing parallel multidimensional index structures.

Table 1. Notations and simulation parameters

Symbol	Definition	Value
T_{comm}	communication time	1.544 Mbps
N_{disk}	number of disks	3 ~ 18
T_{diskIO}	disk I/O time to access a block	1/20,000 second
P_{size}	page size	2 ~ 48 kbyte
N_{server}	number of servers	3 ~ 15
T_{CPU}	CPU time to process a query of a server	
N_{da}	number of disk accesses	
RT	processing time for a range query of a server	$T_{cpu} + N_{da} \times T_{diskIO}$

4.1 PERFORMANCE EVALUATION RESULTS

4.1.1 Performance of the proposed range search and 3 types of k-NN search algorithms

We perform experiments to measure the response time and the disk accesses of k-NN queries and range queries with varying dimensions from 10 to 80, page sizes from 4k ~ 48k and disks from 3 ~ 15. Fig. 7 to 12 show the response time and disk accesses of range searches and three types of k-NN searches. The graph of k-NN type 1 is omitted from the following charts since the performance gap of k-NN type 1 and others is too large to present in the charts with others. We carefully observe the performance of three k-NN queries. From the performance evaluation, we could conclude that our proposed k-NN search algorithms outperform the existing k-NN search algorithm (k-NN type 1). Also, as shown in the Fig.s, the k-NN type 2 out performs slightly the k-NN type 1. The reason is that even though the selectivity of transformed range query in the k-NN type 3 may be smaller than that in the k-NN type 2, k-NN type 3 requires more communication messages and more CPU time to gather and filter results from the servers.

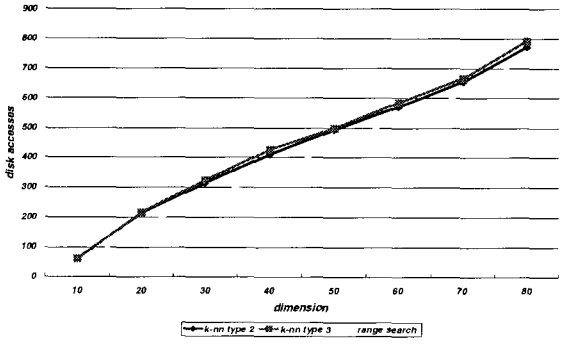


Fig. 7. Number of disk accesses of search operation with varying dimensions (data set:100K, page size:4k, disks:15, servers : 3)

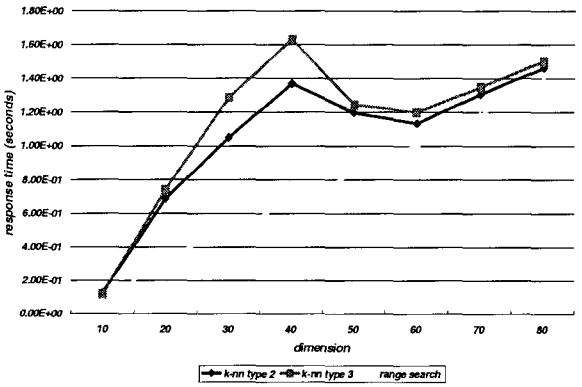


Fig. 8. Response time of search operation with varying dimensions (data set: 100K, page size: 4k, disks:15, servers: 3)

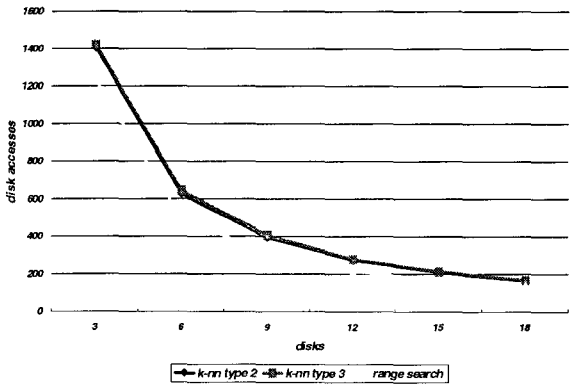


Fig. 9. Disk accesses of search operation with varying the number of disks (data set : 100K, page size : 4k, dimension : 20, servers : 3)

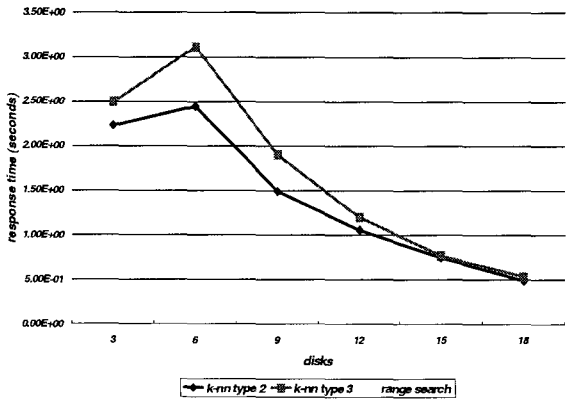


Fig. 10. Disk accesses of search operation with varying the number of disks (data set : 100K, page size : 4k, dimension : 20, servers : 3)

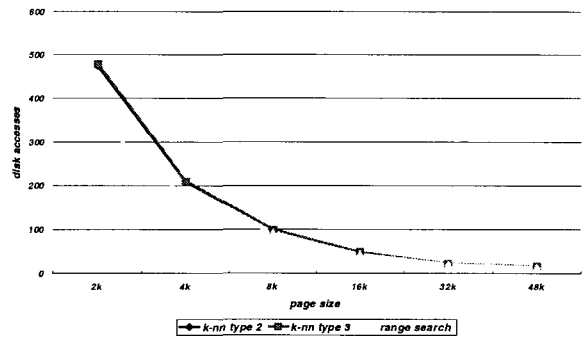


Fig. 11. Disk accesses of search operation with varying page size (data set : 100K, disks : 15, dimension : 20, servers : 3)

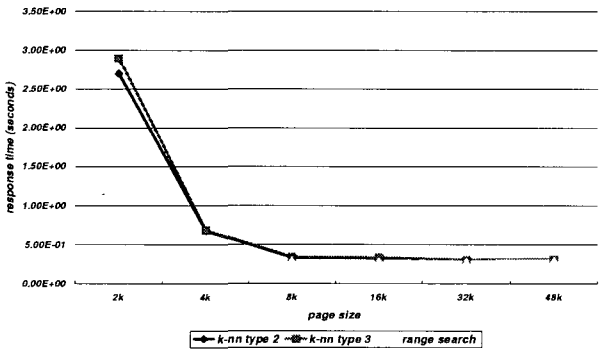


Fig. 12. Response time of search operation with varying page size (data set : 100K, disks : 15, dimension : 20, servers : 3)

4.1.2 Comparison of the performance of range search algorithms

We perform various experiments to measure disk accesses and response time of the range search operations of MCR-tree and the PR-tree with varying the number of disks from 3 to 15. As shown in Fig. 13 and 14, the PR-tree outperforms MCR-tree in all cases. In the MCR-tree, each server and client construct R-trees on one disk. However, we present an architecture that servers builds R-trees on multiple disks. Also, our new range search algorithms improve the disk I/O parallelism.

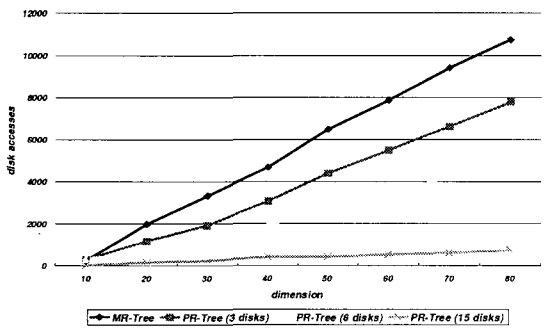


Fig. 13. Disk accesses of search operation with varying dimension (data set : 100K, servers : 3, page size : 4k, disks : 15)

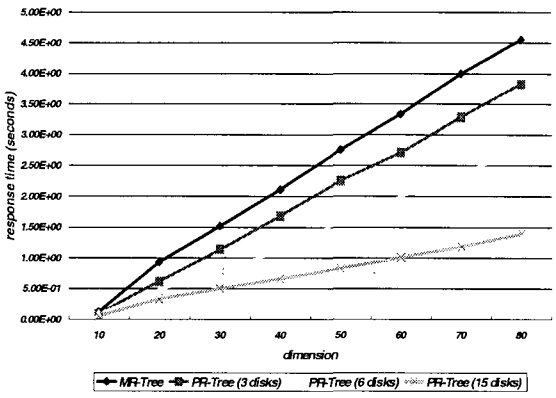


Fig. 14. Response time of search operation with varying dimension (data set : 100K, servers : 3, page size : 4k, disks : 15)

4.1.3 Comparison of the performance of k-NN search algorithms

The Fig. 15 and 16 shows the results of performance comparisons between k-NN search algorithms of PR-trees and MCR-trees. As shown in the Fig.s, PR-trees outperform MCR-trees about 3 times when comparing only k-NN search algorithms. In MCR-trees, there is only one global R-tree that contains only internal nodes and leaf nodes of the global R-tree are organized as R-trees in multiple clients. k-NN search algorithms require searchers to take paths downward and backward repeatedly. Therefore, communication messages between master and clients increase. Also, since our k-NN algorithms is to transform k-NN queries to range-queries, searchers get improved disk I/O parallelism as described in the previous section.

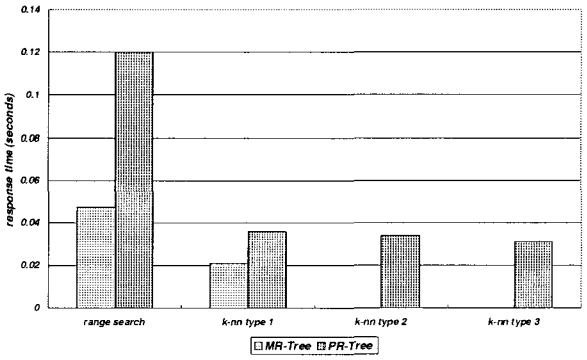


Fig. 15. Response time of search operations (dimension : 9, data set : real 100K, disks : 3, servers : 3)

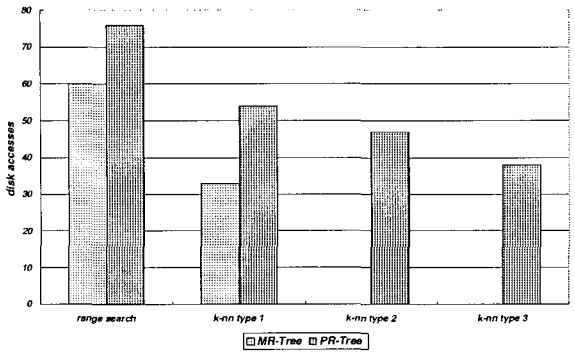


Fig. 16. Response time of search operations (dimension : 9, data set : real 100K, disks : 3, servers : 3)

5. CONCLUSION

In this paper, we proposed an efficient parallel multidimensional index structure. The proposed index structure is nP-n×mD structure that combines 1P-nD structure with nP-nD structure. We present new range search algorithms that more efficiently use disk I/O parallelism. Even though the k-NN search are one of the important query type in multidimensional index structures, researches on improving k-NN search performance in parallel multidimensional index structures are hardly noticed. We present a new k-NN search algorithm that improves the disk I/O parallelism. Through various experiments, we prove that our proposed index structure outperforms exiting parallel multidimensional index structures. In the future, we will implement the proposed index structure based on Storage Area Network which provides shared nothing or shared disk environment and perform various experiments in real parallel computing environment.

REFERENCES

[1] J. Nievergelt, H. Hinterberger, and K. Sevcik, "The grid file: An adaptable, symmetric multikey file structure", ACM Transactions on Database Systems, Vol. 8, No. 1, pp. 38-71, 1984.

[2] J. T. Robinson, "The K-D-B-tree: A search structure for large multidimensional dynamic indexed", Proc. ACM SIGMOD Conference, pp. 10-18, 1981.

[3] M. Freeston, "The BANG file: a new kind of grid file", Proc. VLDB, pp. 260-269, 1987.

[4] A. Guttman, "R-Trees: A dynamic index structure for spatial searching", Proc. ACM SIGMOD Conference, pp. 47-57, 1984.

[5] T. Sellis, N. Roussopoulos and C. Faloutsos, "The R+-Tree: a dynamic index for multidimensional objects", Proc. VLDB, pp. 507-518, 1987.

[6] N. Beckmann, H. P. Kornacker, R. Schneider and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles", Proc. ACM SIGMOD Conference, pp. 322-331, 1990.

[7] S. Berchtold, D. A. Keim and H-P. Kriegel, "The X-tree: An Index Structure for High-Dimensional Data", Proc. VLDB, pp. 28-39, 1996.

[8] N. Katayama and S. Satoh, "The SR-Tree: An index structure for high dimensional nearest neighbor queries", Proc. ACM SIGMOD, pp. 369-380, 1997.

[9] K. Lin, H. V. Jagadish, and C. Faloutsos, "The TV-Tree an index structure for high dimensional data", VLDB Journal, Vol. 3, No. 4, pp. 517-542, 1994.

[10] J. S. Yoo, S. H Lee, K. H. Cho and J. S. Lee, "An Efficient Index Scheme for High-Dimensional Image Data", International Journal of Information Technology, Vol. 6, No. 1, 2000. 5, pp. 1-15

[11] K. Chakrabarti and S. Mehrotra., "The Hybrid Tree: An Index Structure for High-Dimensional Feature Spaces", Proc. ICDE, pp. 440-447, 1999.

[12] R. Weber, H. J. Scheck and S. Blott, "Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," Proc. VLDB, pp. 194-205, 1998.

[13] A. Gionis, P. Indyk and R. Motwani, "Similarity Search in High Dimensions via Hashing", Proc. VLDB, pp. 518-529, 1999.

[14] I. Kamel and C. Faloutsos, "Parallel R-trees", Proc. ACM SIGMOD, pp. 195-204, 1992.

[15] S. Berchtold, C. Bohm, B. Braunmuller, D. A. Keim and H. P. Kriegel, "Fast Parallel Similarity Search in Multimedia Databases", Proc. ACM SIGMOD, pp. 1-12, 1997.

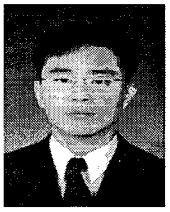
[16] K. S. Bang and H. Lu, "The PML-tree: An Efficient Parallel Spatial Index Structure for Spatial Databases", Proc. ACM Annual Computer Science Conference, pp. 79-88, 1996.

[17] B. Scnnitzer and S. T. Leutenegger, "Master-Client R-trees: A New Parallel R-tree Architecture", Proc. SSDBM, pp. 68-77, 1999.

[18] X. Fu, D. Wang, W. Zheng and M. Sheng, "GPR-Tree : A Global Parallel Index Structure for Multiattribute Declustering on Cluster of Workstations",APDC, pp. 300-306, 1997.

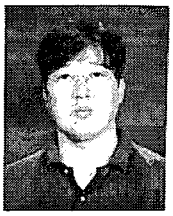
[19] R. Weber, "Parallel VA-File," Proc. ECDL, pp.83-92, 2000.

[20] B. Wang, H. Horinokuchi, K. Kaneko and A. Makinouchi, "Parallel R-tree Search Algorithm on DSVM", Proc. DASFAA, pp. 237-245, 1999.



Kyoung Soo Bok

He received the B.S. in Mathematics from Chungbuk National University, Korea in 1998 and also received M.S. and Ph.D, repectively. in Computer and Communication Engineering from Chungbuk National University, Korea in 2000 and 2005. He is now Postdoc in Korea Advanced Institute of Science and Technology, Korea. His main research interests include location based services, spatio-temporal database, storage management system and content-based retrieval system.



Seok Il Song

He received the B.S. M.S. and Ph.D. in Computer and Communication Engineering from Chungbuk National University, Korea in 1998, 2000 and 2003, respectively. He is now a full-time lecturer in Computer Engineering, Chungju National University, Korea. His main research interests include database system, distribute computing, index structure, location based services and storage management system.

Jae Soo Yoo

He received the B.S. in Computer Engineering from Chonbuk National University, Korea in 1989 and also received M.S. and Ph.D. in Computer Science from Korea Advanced Institute of Science and Technology, Korea in 1991 and 1995. He is now an associate professor in Computer and Communication Engineering, Chungbuk National University, Korea. His main research interests include database system, multimedia database, location based services, distributed computing and storage management system.