Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 Vol I
IMECS 2008, 19-21 March, 2008, Hong Kong

A Scalable Index Mechanism for High-Dimensional Data in Cluster File Systems

Kyu-Woong Lee * Hun-Soon Lee, Mi-Young Lee, Myung-Joon Kim †‡

Abstract—**We address the problem of designing index structures that allow efficient search for approximate nearest neighbors in cluster file systems and especially the scalability of index to parallel execution in order to provide adequate contents-based retrieval upon overwhelming multimedia data. Given large image and video collections that are spread over the world wide web, one of the most challenging field is scalability in a view of performance. Almost existing index methods for high-dimensional data are designed to run on a single machine. In this paper, we propose the hybrid spill tree with local signature files. We describe our scalable index structure and how it can be used to find the nearest neighbors in the cluster environments.**

Keywords:high-dimensional index, feature vector, signature, contents-based retrieval, cluster systems

1 Introduction

The need to manage various types of large scale data stored in web environments has drastically increased and resulted in the development of index mechanism for high-dimensional feature vector data about such a kinds of multimedia data. Recent search engine for the multimedia data in web environment may collect billions of images, text and video data, which makes the performance bottleneck to get a suitable web documents and contents. Given large image and video data collections, a basic problem is to find objects that cover a given information need. Due to the huge amount of data, keyword-based techniques are too expensive, requiring too much manual intervention. In contrast, a content-based information retrieval(CBIR) system identifies the images most similar to a given query image or video clip. It carries out a similarity search.

A common approach to similarity search is to extract features like a color information from the objects. Performing similarity search to find objects most similar to a given object is a classical problem with many practical web applications. These problems involve a collection of various kinds of objects that are characterized by a collection of relevant features. A feature typically is represented by a point in a high-dimensional data space. The number of features that is represented by dimensionality ranges anywhere from tens to thousands. The low-dimensional case is well solved, so the main issue is that of dealing with a large number of dimensions, the so-called "curse of dimensionality"

Thus there is a need of indexing techniques that are able to support execution of similarity queries in high-dimensional space and furthermore these index techniques should have the scalability to be adopted in cluster environment. Despite decades of proposed index methods for high-dimensional data, the current solutions are not entirely satisfactory because they have not considered the scalability. If the system was to index significant fraction of the web, the number of multimedia data to index would be at least of the order billion. From the performance point of view, we need the highly scalable index mechanism which can run on the distributed computing nodes of cluester file systems because the web search engine have to deal with billions of images and video data in the web.

In this paper, we propose the highly scalable index mechanism in cluster file systems, which can be easily extended and executed parallelly on the clustered nodes so that the higher performance can be achieved.

After providing some background and related works in section 2, section 3 proposes the scalable index mechanism in cluster file systems. Our method is designed to parallely executed on the multiple nodes in clustered systems and is generally more suitable for management of billions of image and video data stored in the web. We use a tree-based index structure and signature of feature vector to manage gigantic size of multimedia data. Section 4 concludes and suggests topics for further research.

*Dept. of computer Science, Sangji University, Wonju Korea, leekw@sangji.ac.kr

†Internet Server Group, ETRI, KOREA, {hunsoon, mylee, joonkim}@etri.re.kr

‡This work was supported by the IT R&D program of MIC/IITA. [2007-S-016-01, Development of Cost Effective and Large Scale Global Internet Service Solution]

2 Background and Related Works

There is an underlying needs of indexing techniques which should support execution of similarity queries. A web search engine that provides the contents-based retrieval upon the multimedia data requires complex distance functions to measure similarities of multi-dimensional features, such as shape, color, and motion histogram.

Nearest Neighbor Search Nearest neighbor search has been studied in the last decade. Each object is represented by a feature vector with high-dimension. Given a new object's features, the result is to find the existing object which has the closest feature vector according to designated distance measure. More formally speaking, given a set of n points $P = \{p_1, ..., p_n\}$ in a metric space X with distance function d, preprocess P so as to efficiently answer queries for finding the point in P closest to a query point $q \in X$.

A general approach to the *"similarity indexing "* among the proposed in recent years is an index structure based on *metric tree* [1, 2, 3, 4]. *Metric Trees* which is related to *Ball Trees* are a spatial hierarchical structure for supporting efficient nearest neighbor search. It is a binary tree whose nodes represents a set of points. The root node represents all points, and the points represented by an internal node v is partitioned into two subsets, represented by its two children. Upon these trees, indexing a *metric space* means to provide an efficient support for acquiring the answer *similarity queries*, i.e. queries whose purpose is to retrieve the objects which are similar to a reference query object, and where the similarity between objects is measured by a specific metric distance function d. Formally, $N(v)$ denotes the set of points represented by node v and $v.lc$ and $v.rc$ denotes the left and right child of node v. Then we have $N(v) = N(v.lc) \cup N(v.rc)$ and $\emptyset = N(v.lc) \cap N(v.rc)$ for all the non-leaf nodes. Each node has two pivots denoted as $v.lpv$ and $v.rpv$ so that the distance between them is the largest of all pair distances within $N(v)$ and find the decision boundary L, as shown in figure 1. All points to the left of L belong to $v.lc$ and all points to the right of L belong to $v.rc$. A search on a metric tree is very efficient when the dimension of a data set is low but it start to slow down as the dimension of the data sets increases. Moreover it spends up to 95% of the time in order to verify that it is the true nearest neighbor with backtracking.

Hybrid Spill-Trees A *spill-tree* [5] is a improved version of metric trees in which the children of a node can spill over onto each other and share objects between left and right children. Upon the partition procedure of metric trees, the data points of $v.lc$ and $v.rc$ in a metric tree are disjoint. Unlike metric trees, the spill tree node can share data objects between two children. In a spill tree,

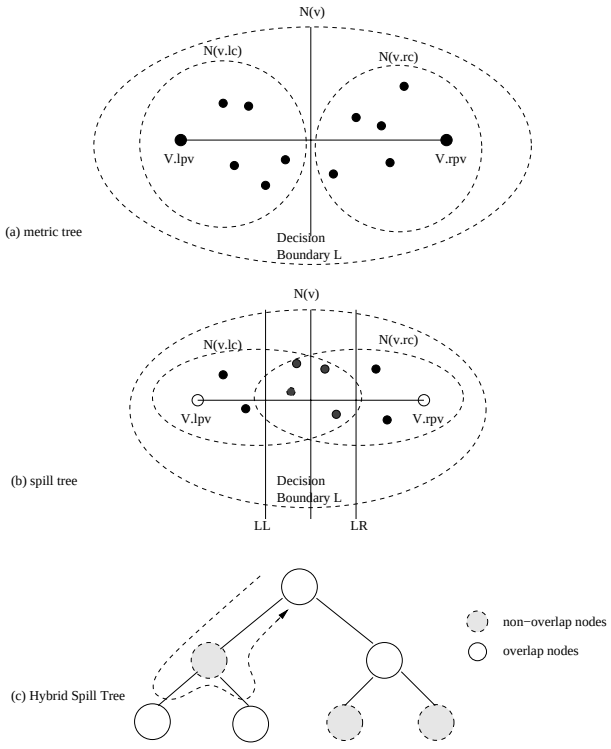


Figure 1: Metric Trees and Hybrid Spill Trees

the splitting criteria is extended to allow overlaps so that some data points may belong to both $v.lc$ and $v.rc$. We choose two pivots $v.lpv$ and $v.rpv$ and determine the decision boundary L like a metric tree. Then we define two new separating boundary, LL and LR, both of which are parallel to L and at distance τ from L. All the points to the right of LL belong to the child $v.rc$, and all the points to the left of LR belong to the child $v.lc$. The region between LL and LR is called *overlapping buffer* and the points fall in this region are shared by $v.lc$ and $v.rc$. The overlap buffer width is 2τ. A spill tree based nearest neighbor search uses defeatist search, which descends the tree quickly using the decision boundaries at each level without backtracking [5].

The depth of spill tree however may vary considerably depending on the overlapping size τ. In the case of $\tau = 0$, a spill tree is the same as a metric tree with depth $O(logn)$. On the other hand, If $\tau \geq \| v.rpv - v.lpv \| /2$, then $N(v.lc) = N(v.rc) = N.(v)$. In other words, both children of node v contain all points of v. In this case, the depth of spill tree is ∞

Hence hybrid spill trees are used which are a combination of spill trees and metric trees. At each node, a decision should be made whether to use an overlap node in a spill

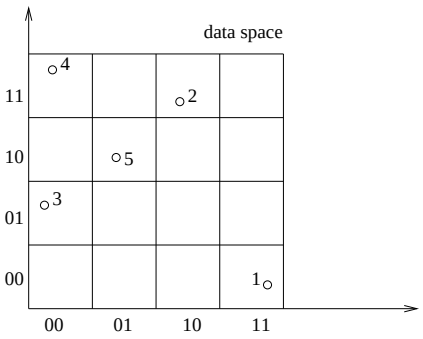
tree or non-overlap node in a metric tree. For non-overlap nodes, the backtracking is needed like a conventional metric tree but we can still use defeatist search on overlap nodes without backtracking. In practice, hybrid spill tree are actually used.

Signature Files Signature indexing methods have been basically developed for text databases [6, 7]. A signature is a small abstraction of an object, which is typically encoded as a bit sequence. Given the signatures of a set of objects and of a query, efficient bit-wise operations can be used to eliminate many objects which cannot possibly match the query. Thus, a signature file is in effect a filtering technique. The *VA-file* [8, 9, 10, 11, 12] is a linear approach that uses approximations of the signature vectors instead of the full feature vectors for most computations. The *VA-file* consists of two separate files: the vector file containing the feature data, and the approximation file containing a quantization(*signature*) of each feature vector. The nearest neighbor of a query is found similarly to signature techniques [6, 7]. In *VA-file* structure, for each vector, a b-bit approximation is derived. Each dimension j has the number of bits b_j. A first step scans only the signature file and then a second step accesses some of the vectors corresponding to signature candidates to determine the nearest neighbor. The relationship between the number of bits(b) and the number of dimensions(d) determines b_j as follows.

$$b_j = \lfloor \frac{b}{d} \rfloor + \begin{cases} 1, j \leq b \mod d \\ 0, \quad otherwise \end{cases}$$

The figure 2 shows an example of signature from the real feature data. The point 1 with vector data $\{0.9, 0.1\}$ in figure 2 has a 11 as a X coordinate value and a 00 as a Y coordinate value. Thus the signature of point 1 is 1100. The major advantage of signature indexing method is that it retains good performance as dimensionality increases and is relatively to parallelize and distribute. However it still has the difficulty to avoid the sequential search in the signature file. Even though a signature file can be stored in the distributed manner, the entire set of partitioned signature file should examined in order to get the k nearest neighbor of a given query.

Hybrid Spill Trees in Parallel The scalable version of an approximate nearest neighbor search algorithm was suggested in the paper [13]. The main challenge in scaling up the hybrid spill tree generation algorithm is that it requires all the object's feature vectors to be in memory. In a collection of over a billion images in web documents, there are nearly a thousand times as many as can fit into one machine's memory. The paper [13] suggested the intelligent partition of the data. They first create a random sample of data small enough to fit on a single



	Feature Vector		Signature
1	0.9	0.1	11 00
2	0.6	0.8	10 11
3	0.1	0.4	00 01
4	01.	0.9	00 11
5	0.3	0.7	01 10

Figure 2: An Example of Signature File

node, say $1/M$ of the data, and build the *metric tree* for this data. Each of the leaf node in this *top metric tree* then defines a partition, for which a *hybrid spill tree* can be built on a separate machine, as described in figure 3.

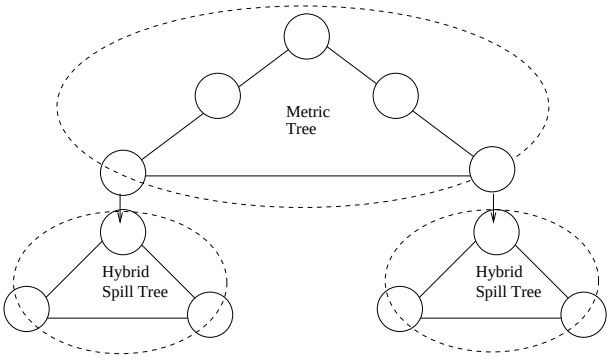


Figure 3: Hybrid Spill Trees in Parallel

3 Hybrid Spill Tree with Signature files

For the nearest neighbor search, the tree-based and signature-based indexing methods have been proposed. The tree-based indexing still has the difficulty since we have to perform the backtracking in a tree in order to get the k nearest neighbor. It is not convenient to extend to the distributed or cluster file systems. The signature-

based indexing has also disadvantages because of scalability into the cluster file systems. Moreover very large scale image collections are difficult to efficiently organize and navigate in a machine.

Thus, we propose a scalable indexing mechanism for finding the approximate nearest neighbor in the billions of images and other multimedia data. Our method use a basic concept of hybrid spill tree and signature file. First the hybrid spill tree is built based on the sample vector data that is extracted from the original feature vector file. The size of sample feature data has to be determined by the capacity of a machine that accommodate the hybrid spill tree. All the leaf node of hybrid spill tree makes the signature file for feature vectors of the corresponding range of each node in hybrid spill tree.

3.1 Overall Construction Procedure

The high dimensional data space is transformed into a hybrid spill tree in our method. In this construction step, the hybrid spill tree just consider sampled feature data from the original high dimensional data space. The leaf node which should be accommodated in a distinct machine has a local signature file for the corresponding range of original feature vector. Thus we can support the parallel search through the leaf nodes and ensure the efficient response for the contents-based retrieval.

The figure 4 describes the overall construction procedure of hybrid spill tree with signature files. The *sampler* in

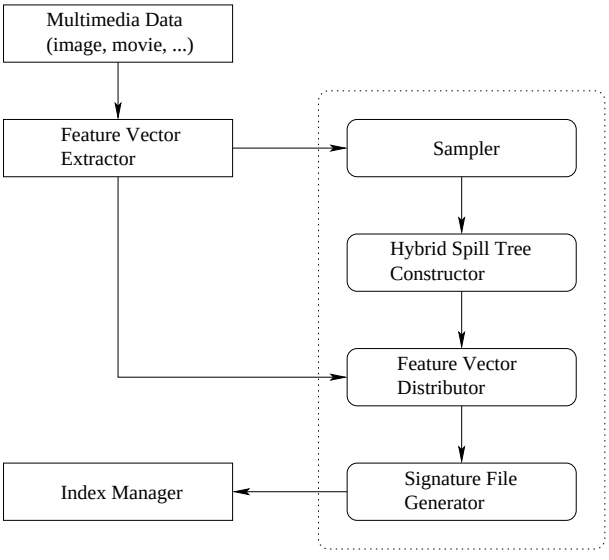


Figure 4: Overall Construction Procedure

figure 4 extracts some feature vector from the original vector file as described at step (1) in figure 5. The sampled feature vectors should be in a machine's memory. Then *hybrid spill tree constructor* makes a basic hybrid spill tree based on the sampled feature vector as shown at step (2) in figure 5. At step (3), the *feature vector distributor* sends the feature vector to the corresponding computing machine according to the range of each leaf node. The *signature file generator* at each machine collects the feature vector which is sent by *distributor* and makes its signature and stores in its own local signature file, as illustrated in figure 5.

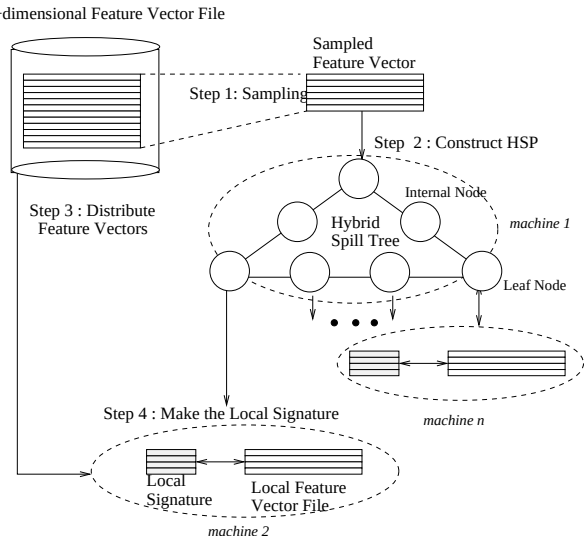


Figure 5: Hybrid Spill Tree with Local Signature Files

3.2 Query Processing in Hybrid Spill Tree with Local Signature Files

After the hybrid spill trees with signature files have been built, they can be queried. The top tree with the signature files can be regarded as one large hybrid spill tree which is managed in a machine. As mentioned in previous section, the top hybrid spill tree cannot consider all feature vector data due to the capacity of a machine's memory. In order to find the k nearest neighbor, the top tree only guide the way which node(machine) has the the nearest neighbor from the given query point. At this step, if the parent node of leaf node is not a overlapped node, the candidate node can be more than one node because of the basic characteristics of hybrid spill trees. We hence the query send to multiple candidate nodes and each node process it to find the nearest neighbor concurrently. At each node, the local signature file is used to find the nearest neighbor. For the signature of the given query, the local signature file is scanned sequentially.

Figure 6 describes the overall process of query processing in hybrid spill tree with local signature files and the traverse for the nearest neighbor is illustrated in figure 7 For the given query, we first extract the feature vector of the query by using conventional extracting tools. Then

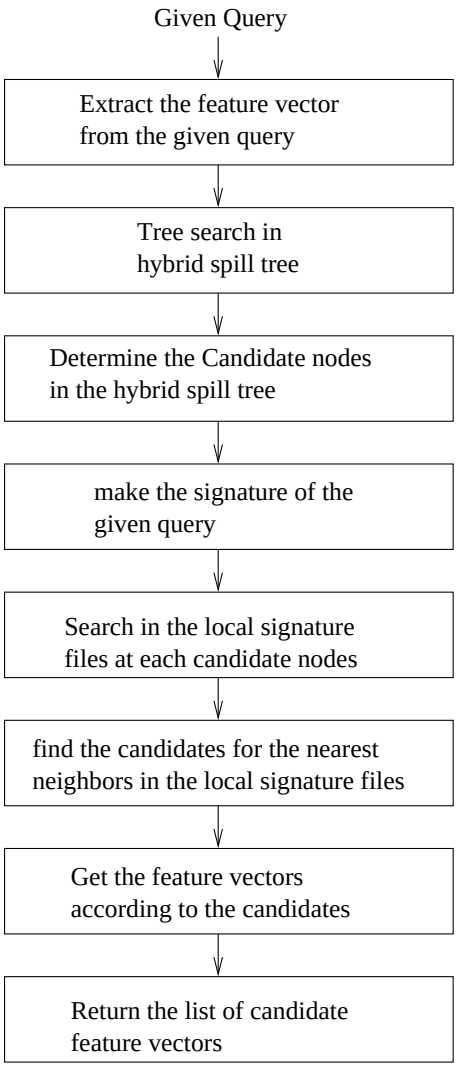


Figure 6: Query Processing in Hybrid Spill Tree with Local Signature Files

we traverse the top tree in order to determine which leaf nodes can process the given feature vector. After the candidates are determined, the candidates make the signature of the query's feature vector and search in the local signature file at each node. We finally decide the feature vector as the result of query in the local feature vector files by using the candidates of the local signature file.

Our proposed hybrid spill tree with local signature files ensures the efficient processing to find the nearest neighbor since our index structure can be easily partitioned to cluster environments and operated by multiple computing nodes.

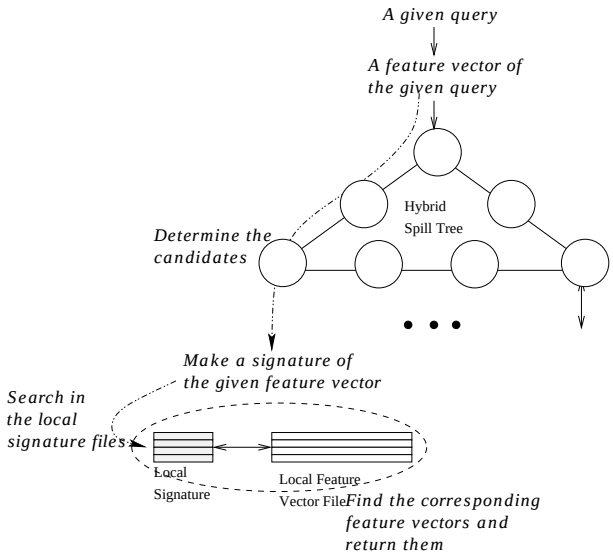


Figure 7: Traverse in Hybrid Spill Tree with Local Signature Files

4 Conclusion

Almost all existing index methods for high-dimensional data are designed to run on a single machine and are not useful to extend to cluster environments. Given large image and video collections that are spread over the world wide web, we have to provide the index scalability in a view of performance. Thus, in this paper, we have proposed the highly scalable index mechanism which can run on the distributed computing nodes of cluester file systems because the web search engine has to deal with billions of images and video data in the web. We have described the hybrid spill tree with local signature files that can be used for building parallel distributed index structure.

References

[1] P. Ciaccia, M. Patella, and P. Zezula, "M-tree : An efficient access method for similarity search in metric spaces," in *Proc. of the 23rd VLDB Conference Athens, Greece*, 1997.

[2] P. Ciaccia, P. Zezula, and M. Patella, "M-tree: An efficient access method for similarity search in metric spaces," in *The VLDB Journal*, 1997, pp. 426–435. [Online]. Available: citeseer.ist.psu.edu/article/ciaccia97mtree.html

[3] P. Ciaccia and M. Patella, "Bulk loading the m-tree," in *Proc. of the 9th Australasian Database Conference (ADC'98), pages 15–26, Perth, Aus-*

tralia, February 1998., 1998. [Online]. Available: citeseer.ist.psu.edu/ciaccia98bulk.html

[4] A. W. Moore, "The anchors hierarchy: Using the triangle inequality to survive high dimensional data," in *Proc. of the 20th Conference on Uncertainty in Artificial Intelligence*, 2000, pp. 397–405. [Online]. Available: citeseer.ist.psu.edu/moore00anchors.html

[5] T. Liu, A. W. Moore, A. Gray, and K. Yang, "An investigation of practical approximate nearest neighbor algorithms," in *Proc. of the Adavances in Neural Information Processing Systems*, 2004.

[6] Z. Lin and C. Faloutsos, "Frame-sliced signature files," *IEEE Transaction on Data Engienering*, vol. 4, no. 3, 1992.

[7] Z. Lin, "Concurrent frame signature files," *Distributed and Parallel Databases : An International Journal*, vol. 1, no. 2, 1993.

[8] R. Weber and S. Blott, "An approximation based data structure for similarity search," 1997. [Online]. Available: citeseer.ist.psu.edu/weber97approximationbased.html

[9] R. Weber, K. Böhm, and H.-J. Schek, "Interactive-time similarity search for large image collections using parallel VA-files," in *Lecture Notes in Computer Science*, vol. 1923, 2000, pp. 83+. [Online]. Available: citeseer.ist.psu.edu/344285.html

[10] S. Blott and R. Weber, "A simple vector approximation file for similarity search in high-dimensional vector spaces," 1997. [Online]. Available: citeseer.ist.psu.edu/blott97simple.html

[11] R. Weber and K. Böhm, "Trading quality for time with nearest-neighbor search," *Lecture Notes in Computer Science*, vol. 1777, 2000. [Online]. Available: citeseer.ist.psu.edu/weber00trading.html

[12] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, 1998, pp. 194–205. [Online]. Available: citeseer.ist.psu.edu/weber98quantitative.html

[13] T. Liu, C. Ronsenberg, and H. A. Rowley, "Clustering billions of images with large scale nearest neighbor search," in *Proc. of the IEEE Workshop on Applications of Computer Vision*, 2007.