Query Optimization in Object Oriented Databases Based on Signature File Hierarchy and SD-Tree

Tran Minh Bao(✉) and Truong Cong Tuan

College of Science, Hue University, 77 Nguyen Hue Street, Hue City, Viet Nam
tmbaovn@gmail.com, tctuan_it_dept@yahoo.com

Abstract. Direct query on objects in object-oriented databases costs a lot of data storage during query processing and time to execute query on real data systems. Recently, there are many researches focusing on resolving that problem by indexing on single classes, class hierarchies or nested objects hierarchies. In this paper, we propose a new indexing approach. This approach is based on the technique of using signature files and SD-Trees where signature files are in hierarchical organization to quickly filter irrelevant data and each signature file is stored in the similar structure with SD-Tree to fasten signatures scanning. This technique helps reduce significantly searching space, hence improves significantly time complexity of query.

Keywords: Object-oriented database system · Index · Signature file · SD-Tree · Object-oriented query

1 Introduction

Direct query on objects in object-oriented databases costs a lot of data storage during processing query and time to execute query on real data system. The problem is to describe data system in a more simple way and construct a corresponding data structure to reduce searching space during executing query while necessary objects are ensured to be searched.

To reduce space of data query, proposed indexing techniques used to evaluate query in databases [6] have been developed based on binary tree balancing mechanism which was added some special characteristics to reduce tree balance or minimize accesses to data files. These techniques have been developed to increase query speed in object-oriented databases [10–12]. The main idea is that each SD-Tree on a class in hierarchy is remained but indexes are nested by relation of subclass–target class. Besides indexes in inherited hierarchy structure, many indexing approaches used for nested characteristic query have been proposed [1–3, 7, 9]. Instead of concentrating on inherited hierarchy of classes, researchers have discovered general hierarchy of classes and proposed different index structures following nested characteristics [1, 2, 7, 9] … Signature file storage structures will reduce searching space and optimize data query process.

P.C. Vinh and V. Alagar (Eds.): ICCASA 2015, LNICST 165, pp. 309–321, 2016.
DOI: 10.1007/978-3-319-29236-6_30

It is necessary to construct a data structure for signature file storage to improve searching. These signature file storage structures can be in form of sequential signature files, sliced signature files, signature tree structure, signature graph structure… where the cost of sliced signature file storage is double of sequential signature files and triple of sequential signature files or more [8]. The main advantage of this approach is its effect in processing new insert and query to parts of word. However, when comparing with indexing based on tree structure, using sequential signature files has 2 disadvantages: (1) they cannot be used to evaluate range query; (2) for each processed query, entire signature files need to be scanned, it makes I/O processing cost increase.

In this paper, we try to improve the second problem to a certain point. Firstly, we organize sequential signature files in hierarchical structure to reduce searching space during query evaluating process. Next, we store signature files in form of a SD-Tree to execute scanning only one single signature file. If signature file size is large, time saved by this approach is really significant. In fact, this is a B^+-tree constructed by signature files. Therefore, it can speed up the process of identifying signature position in a signature file. However, in a signature tree, each path is corresponding with one signature identification which can be used to determine its only corresponding signature in signature file. This way helps quickly find out a set of corresponding signatures with query signature.

The remaining of this paper is presented as follows. In Part 2, we provide background. Part 3 proposes an approach combining signature files and SD-Tree hierarchy. Finally, Part 4 gives the conclusion.

2 Background

2.1 Characteristic Signature

In an object-oriented database, each object is presented by a set of characteristic values. Signature of an characteristic value is a sequence of hashed-code bits. Given an characteristic value, for example the word “student”, we decompose it into a string of three-letter sets as follow: “stu”, “tud”, “ude”, “den” and “ent”. Then, using hash function h, we map a triplet to an integer k which means kth bit in a string assigned value 1. For example, assuming that we have h(stu) = 2, h(tud) = 7, h(ude) = 10, h (den) = 5 and h(ent) = 11. Then we create a bit string: 010 010 100 110 which is signature of the word.

2.2 Characteristic Signature, Signature File

Object signature is constructed by logical OR algorithm for all signatures of characteristic values of the object. Below is an example of an characteristic signature:

Example 1. Consider an object which has characteristic values of “student”, “12345678”, “professor”. Suppose that signature of these characteristic is:

010	010	100	110
100	010	010	100
110	100	011	000

In this case, object signature is 110 110 111 110, generated from characteristic signatures by using logical OR algorithm. Object signatures of a class are stored in a file, called object signature file.

2.3 Query Signature

An object query will be encoded into a query signature together with hash function applied to objects. When a query needs to be executed, object signatures will be scanned and unmatched objects will be excluded. Then query signature is compared with object signatures of signature file. There are three possibilities:

(i) The object matches with the query, i.e., for every bit in query signature s_q, corresponding bit in object signature s is the same, i.e., $s_q \wedge s = s_q$, a real object of query.
(ii) The object does not match with the query, i.e., $s_q \wedge s \neq s_q$.;
(iii) Signatures are compared and matching one is found but its object does not match with searching condition of the query. To eliminate this case, objects must be checked after object signatures are matched.

Example 2. This example illustrates the query for object signature in Example 1:

Query :	Query signature :				Result :
student	010	000	100	110	successful
john	011	000	100	100	unsuccessful
11223344	110	100	100	000	false drop

Comment: comparing query signature s_q to object signature s is incorrect comparison. That means, query signature s_q matches with signature s if for any 1 bit in s_q, the corresponding bit in s is also 1 bit. However, for any 0 bit in s_q, the corresponding bit in s can be 0 or 1.

2.4 Querying Object-Oriented Databases

In object-oriented CSDL system, an entity displayed according to object type including methods and properties. Objects have similar methods and properties gathered in the same layer. If the *C* layer has a complex property with domain *C'*, so we shall create relation between *C* and *C'*. This relation is a general relation. When using arrows to connect layers for displaying general relation, have to create general hierarchies for displaying nested structure of layers.

Example 3. An example for nested object hierarchy system illustrated such as follows:

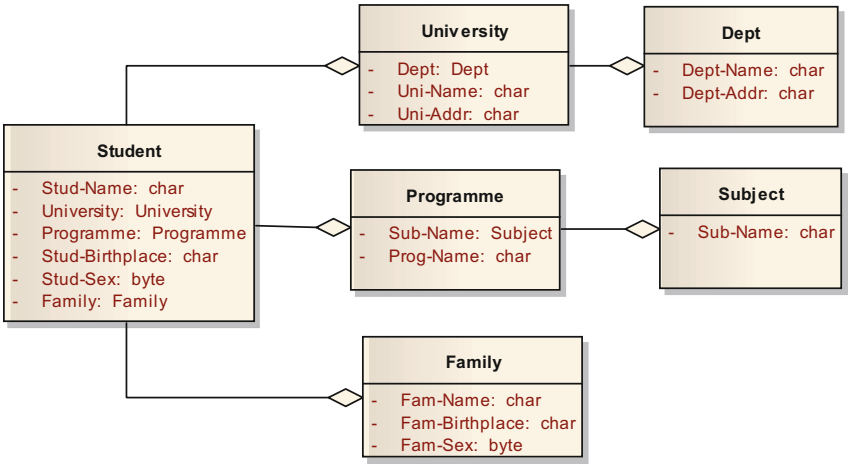

Fig. 1. An example of a nested object hierarchy

Object *o* referenced is a property of object *o'*, then object *o* considered as nested in *o'*, and *o'* considered as 'father-object' of *o*.

In object-oriented CSDL system, condition found in query collected in a properties collection. This property is nested property of target layers.

Example 4. The query "retrieve all students born in *Ben Tre* of dept *information technology*" can be expressed as:

```
Select Student
Where Student.Stud-Birthplace = "Ben Tre"
And   Student.University.Dept.Dept-Name   =   "information
technology"
```

Without indexing structures, the above query can be evaluated in a top-down manner as follows. First, the system has to retrieve all of the objects in the class Student and single out those who were born in *Ben Tre*. Then, the system retrieves the University objects referenced by the Student born in *Ben Tre* and checks the Dept-Name of the Dept. Finally, those Students born in *Ben Tre* by a University that has Dept *information technology* are returned.

2.5 Signature File Hierarchy and Query Algorithm

2.5.1 Signature File Hierarchy

Purpose of using signature file: remove unconditional objects, means if a signature is not suitable with query signature so the object related with this signature surely ignored. So therefore we do not need to access to these objects.

Example 5. Signature and signature file hierarchy:

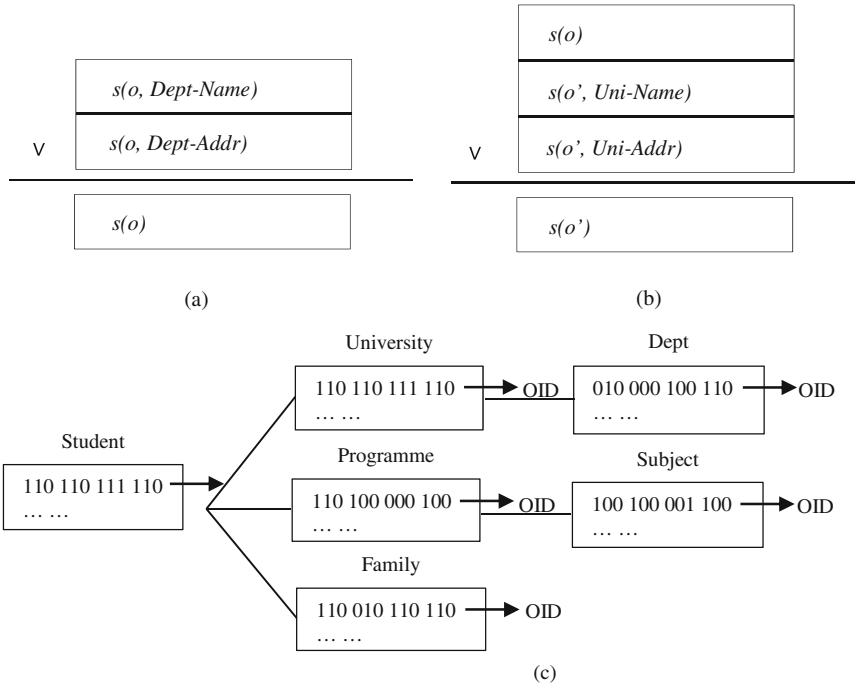

Fig. 2. Signature and signature file hierarchy

Considering Dept layer in *0* hierarchy of complex properties in Fig. 1. Signature of *o* object can be created by method in Fig. 2(a), each *s(o, x)* signature symbol created for property value *x* of *o* and *s(o)* signature symbol *o*. To layers of complex properties, signature of objects can be created with the same method, like layer of original properties. Difference: signature of complex property is signature of referenced object illustrated in Fig. 2(b). In Fig. 2(b), *o'* marked object of University layer. And *o* object of Dept layer is property value of Dept of *o'*. Hierarchy of signature file can be used for building database displayed in Fig. 1 also illustrated in Fig. 2(c).

2.5.2 Query Algorithm Based on Signature File

Using query signature tree to decrease searching space. This method, we need two *stack* structures to control prioritize scanning according to depth of tree structures: $stack_q$ to *Q(s, t)* and $stack_c$ to class hierarchy. In $stack_q$, each component is a signature, meanwhile in $stack_c$, each component is a collection of objects belong to the same layer can be approached by scanning class hierarchy.

Algorithm 1. [4] top-down-hierarchy-retrieval;

Input: an object query Q;

Output: a set of OIDs whose texts satisfy the query.

Method:

Step 1. Compute the query signature hierarchy $Q_{(s,t)}$ for the query Q.

Step 2. Push the root signature of $Q_{(s,t)}$ into $stack_q$; push the set of object OID of the target class into $stack_c$.

Step 3. If $stack_q$ is not empty, $s_q \leftarrow$pop $stack_q$; else go to (7).

Step 4. $S \leftarrow$pop $stack_c$; for each oid_i E S, if its signature $osig_i$ does not compare s_q, remove it from S; put S in S_{result}.

Step 5. Let C be the class to which the objects of S belong; let C_1, ..., C_k be the subclasses of C; then partition the OID set of the objects referenced by the objects of S into S_1, ..., S_k such that S_i belongs to C_i; push S_1, ..., S_k into $stack_c$; push the child nodes of s_q into $stack_q$.

Step 6.Go to (3).

Step 7. For each leaf object, check false drops.

This technique helps for optimization when implementing step (4). In this step, some objects selected by using corresponding signature in query signature tree. In step (5), referenced objects and son node's signatures of query signature tree is added to $stack_c$ and $stack_q$. In step (7), conduct inspection on errors.

Example 6. Assuming a part of signature file hierarchy created for a CSDL according to a diagram in Fig. 1 belongs to type described in Fig. 3:

When the first signatures of signature file for Student suitable with signature in query signature tree, signatures are referenced by themselves in signature file for University need to have additional inspection. Assuming the first signature of University is referenced by the first signature in Student meanwhile the second signature in University is referenced by the second signature in Student. We can see that the second signature in University is not suitable with corresponding signature in query signature tree. Therefore all signatures of Dept object is referenced by Dept object won't be inspected (watching grey illustration in Fig. 3). This method is optimal method when comparing with "searching from top to bottom" because in "searching from top to bottom" must inspect all object signatures of Dept.

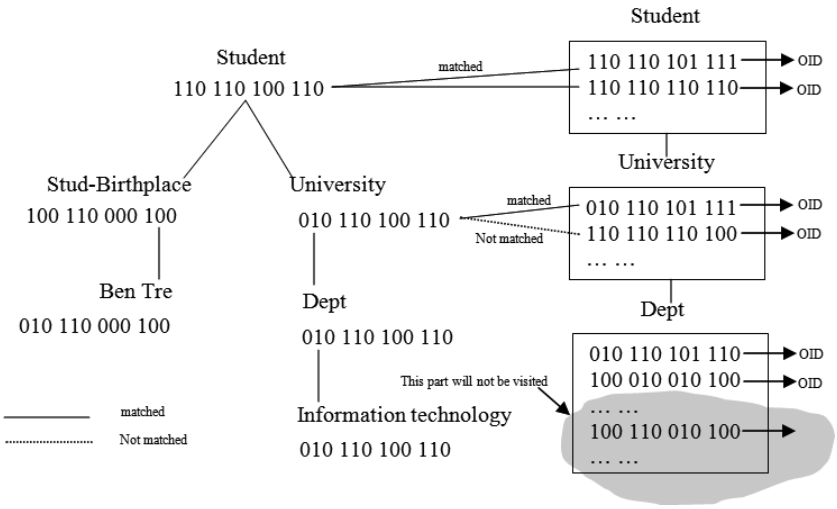

Fig. 3. Illustration of query evaluation

2.6 SD-Tree

2.6.1 Overall Structure of SD-Tree

Technique on creating index in Object-Oriented CSDL system using dynamic balance method of B^+-tree called SD-Tree (Signature Declustering). In this implementation, positions of bit 1 in signature is overall via collection of leaf node. Using this method for an available query signature, so all matched signatures can be queried accumulated in a single-node. Querying, optimal searching method is considered to boost speed of total progress.

Example 7. Overall structure of SD-Tree (Fig. 4):

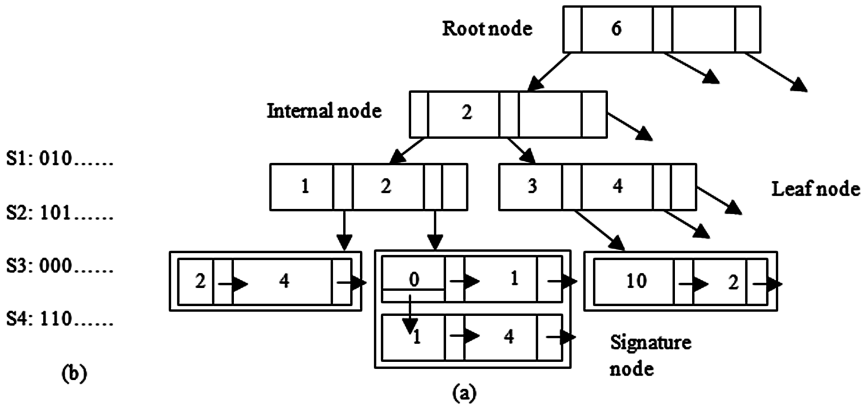

Fig. 4. Overall structure of SD-Tree [13]

Processing a query signature S_q, final appearance of bit 1 at position i in S_q is found out together with creating intermediate prefix (B). Then signature node of leaf node i accessed from root and all signatures with queried prefix B.

Example 8. Give S_q = 011001000110. To search all matched signatures S_q, SD-Tree considered well from root and node values compared with bit's position of S_q. Final appearance of 1 in S_q stay at position 11. Binary prefixes is created for S_q by using position of bit 1 such 0110010001. A node with key-value is 11 accessed in saved signature list bit 1 with collection type and all signatures in signatures list are inspected prefix value 0110010001. Therefore, except bit sample of S_q, all matched signatures are returned in a single-access.

2.6.2 Query Algorithm Based on SD-Tree

Algorithm 2. [14] Search(S_q)

Input: The (query) signature to search.

Output: The list of signatures matching the given signature.

Method:
Step 1. Compute the Signature weight for the query signature.

Step 2. If signature weight is greater than 50% then search the query signature in the leaf nodes for the unset bits.

Step 3. Else search the query signature in the leaf nodes for the set bits.

Step 4. Access leaf node.

Step 5. Compare the prefix of Sq.

Step 6. If Found () then read and output the list of signatures.

Step 7. Else report "no matching signatures".

3 Approach Combining Signature File Hierarchy and SD-Tree

3.1 Query Data Structure Model

Direct query on objects in object-oriented databases costs a large space for data storage during query process and a long time to execute query on real databases. To improve this problem, we need to represent data system more simply and construct corresponding data structure to reduce searching space during query executing process while necessary objects are still retrieved by using signature tree. From [4], to optimize the

query we need to combine signature file hierarchy with signature tree. This has been shown to improve query time. From [13], query time complexity on SD-Tree is much smaller than signature tree's query time complexity. Therefore, we still use signature file hierarchy as in [4] but replace signature tree with SD-Tree to improve query time. Base on theory and suggested algorithms, this paper proposes an approach which combines signature file hierarchy with SD-Tree as follows: (1) all of signature files are organized in hierarchical structure to make it easier for executing stepwise filtering technique; (2) each signature file is stored in form of SD-Tree structure to speed up signature file scanning.

In an object-oriented database, each object is presented by a set of characteristic values. Signature of an characteristic is a string of hash-encoded bits. Object signature is constructed by overlapping all of characteristic signatures of the object. Object signatures of a class are stored in a file, called signature file. Signature files form SD-Tree.

Example 9. Construction of SD-Tree is illustrated as below (Fig. 5):

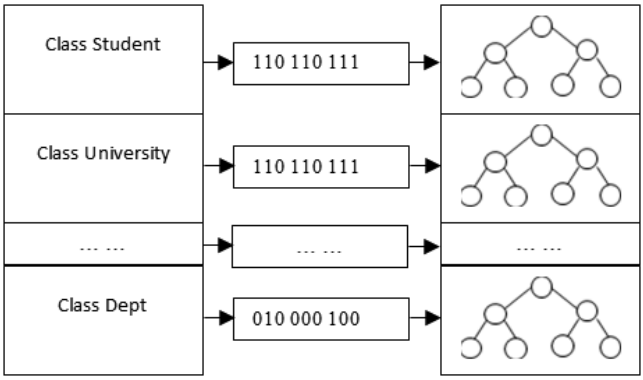

Fig. 5. SD-Tree construction

On an object-oriented database, if a class *C* has an characteristic that is composite with domain *C'*, relation between *C* and *C'* will be created. This relation is called general relation. When connecting these classes by using arrows to present general relation, a general hierarchy is built to present nested structure of classes. Classes are encoded into signature files and signature files form signature file hierarchy. Each signature file forms a SD-Tree.

Example 10. Combination of signature file hierarchy and SD-Tree is illustrated as follow (Fig. 6):

Data structure is stored entirely in the main memory. In this case, inserting and deleting a signature on SD-Tree is executed easily. However, files in databases are usually very big. Therefore, data structure cannot be stored in the main memory but external memory. For object-oriented databases, they will be stored and executed in external memory. An object-oriented database has many classes, each class has many objects. A SD-Tree structure will be constructed corresponding with each class, in the

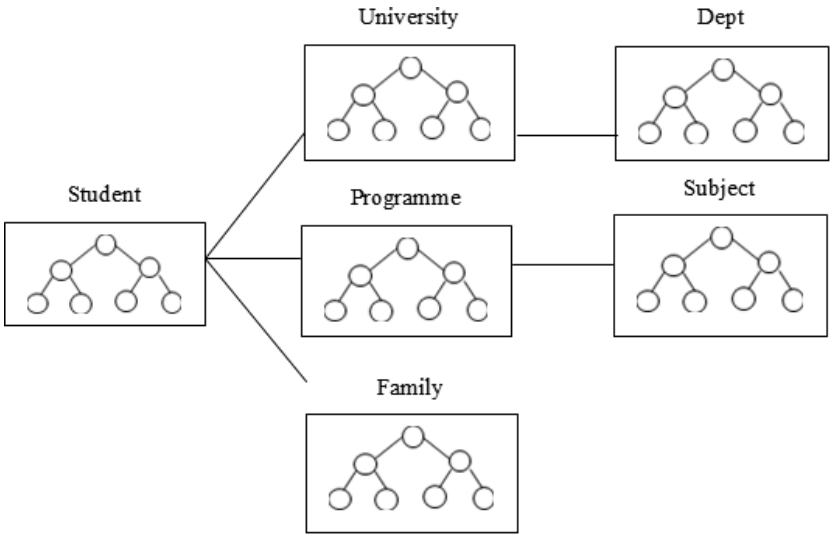

Fig. 6. Signature files hierarchy and SD-Tree

same time, each object will form an object signature. The entire object-oriented database will be organized in form of hash table structure including object signatures to execute queries.

3.2 Object-Oriented Query Processing

To execute a query of an object in an object-oriented database, firstly we have to change an object-oriented database into data structure as above. We do:

```
Step 1. Attribute of the object is hashed into binary
signatures and attributes which form object signatures.
Step 2. Object signatures in a same layer will form SD-
tree.
Step 3. Create signature file hierarchy where each file
is a SD-tree.
```

After having data structure for query, we execute object query process on object-oriented databases as follow:

```
Step 1. Encode key words which need to be retrieved into
binary signature.
Step 2. Execute key word signature query to determine
classes which need to be searched.
Step 3. Execute key word signature query on SD-tree
corresponding with determined classes.
```

3.3 Time Complexity

3.3.1 Comparison of Searching Between Young's Method and Signature File Hierarchy

In [4], to estimate objects accessed in a query using two different methods: (1) Yong method is recommended in [15]; (2) searching according to hierarchy from top to bottom.

(i) *Yong's method*

Yong method, signature of the referenced object will be saved in referenced objects. Then, we can inspect first-order logic on signatures of them before accessing. In this way, we do not need to implement too much mathematical methods I/O.

(ii) *Top down hierarchy retrieval*

This method helps us to select stronger than Yong's method. Because of inspection on a node in query signature hierarchy, not only first-order logic related to current node but also other first-order logics, their effects will be added into links leading to that node. Using query signature hierarchy, has objects in target layer will be removed by inspecting matched signature files, helps to decrease effectively accessed objects.

In [4], we can see that we can get high performance by using hierarchy search method from top to bottom from abstract perspective, query signature hierarchy is a "*general*" filter meanwhile cloning techniques developed by Yong is a "*internal*" filter. These two methods help to decrease accessed objects.

3.3.2 Comparison of Time Complexity Between Signature Tree and SD-Tree

(i) *Signature tree method*

In [13], complexity of time to insert into signature tree is O(nF), n is amount of signatures of files and F is length of signature including bit 0 and bit 1. To signature tree, height of signature tree is limited: $O(\log_2 n)$, n is amount of leaf node. Costs used for searching signature tree normally is $O(\lambda.\log_2 n)$, λ is amount of ways passed.

(ii) *SD-Tree method*

In [13], SD-Tree used according to indexing structure for big data collection, F value is small, time used for creating SD-Tree will be decreased. Complexity of time to insert is limited: O(n.m), n is amount of signatures in files and m is amount of bit 1 in available signature. Another useful characteristic of SD-Tree: with higher F value, by changing p value, h value, height of signature tree can be remained in low-level to boost speed of searching faster limited is O(logp(F/p-1)). Search time is used for a query with a collection of bit at i position, finally is total accessed time on leaf node (T_{li}) and time is used for searching signature node (T_{si}) is calculated like this:

$$T_s = T_{li} + T_{si}.$$

We can see that T_{li} doesn't change for all leaf nodes for a dynamic balance structure such as SD-Tree and T_{si} will be increased when the value of i is increased. Therefore, search time is limited is $O(T_{li} + 2^{i-1})$.

Comparing search time's complexity of signature tree is $O(\lambda.\log_2 n)$ and SD-Tree is $O(T_{li} + 2^{i-1})$, we can see that T_{li} value is very small comparing with λ value, that's a good point of SD-Tree.

4 Conclusion

In this paper, we propose a new indexing technique. This approach is a combination of signature file and SD-Tree hierarchy. To optimize scanning objects hierarchy, we base on signature file hierarchy to reduce number of sub-trees. However, because signature file only works as an incorrect filter, it cannot be ordered or binary searched, thus cannot be used to speed up signature scanning process. Hence, we propose construction of a SD-Tree on the file where signature appears as a node of signature file hierarchy. This technique can avoid sequential searching, thus help reduce time needed for searching on signature file.

References

1. Bertino, E.: Optimization of queries using nested indices. In: Proceedings of International Conference on Extending Database Technology, pp. 44–59 (1990)
2. Bertino, E., Guglielmani, C.: Optimization of object-oriented queries using path indices. In: 2nd International Workshop on Research Issues on Data Engineering: Transaction and Query Processing, pp. 140–149 (1992)
3. Choenni, S., Bertino, E., Blanken, H.M., Chang, T.: On the selection of optimal index configuration in OO databases. In: Proceedings of 10th International Conference on Data Engineering, pp. 526–537 (1994)
4. Chen, Y.: Building signature trees into OODBs. J. Inf. Sci. Eng. **20**(2), 275–304 (2004)
5. Dervos, D., Manolopoulos, Y., Linardis, P.: Comparison of signature file models with superimposed coding. J. Inf. Proc. Lett. **65**, 101–106 (1998)
6. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. Benjamin Cumming, California (1989)
7. Fotouhi, F., Lee, T.G., Grosky, W.I.: The generalized index model for object-oriented database systems. In: 10th Annual International Phoenix Conference on Computers and Communication, pp. 302–308 (1991)
8. Ishikawa, Y., Kitagawa, H., Ohbo, N.: Evaluation of signature files as set access facilities in OODBs. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 247–256 (1993)
9. Kim, W., Kim, K.C., Dale, A.: Indexing Techniques for Object Oriented Databases, pp. 371–394. Addison Wesley, Reading (1989)
10. Kemper, A., Moerkotte, G.: Access support relations: an indexing method for object bases. Inf. Syst. **17**, 117–145 (1992)

11. Low, C.C., Ooi, B.C., Lu, H.: H-trees: a dynamic associative search index for OODB. In: Proceedings of 1992 ACM SIGMOD Conference on the Management of Data, pp. 134–143 (1992)
12. Sreenath, B., Seshadri, S.: The hcC-tree: an efficient index structure for object oriented database. In: Proceedings of International Conference on Very Large Database, pp. 203–213 (1994)
13. Shanthi, I.E., Nadarajan, R.: Applying SD-tree for object-oriented query processing. Informatica (Slovenia) **33**(2), 169–179 (2009)
14. Thakur, A., Chauhan, M.: Optimizing search for fast query retrieval in object oriented databases using signature declustering. Int. J. Eng. Res. Dev. 46–50 (2012)
15. Yong, S., Lee, S., Kim, H.J.: Applying signatures for forward traversal query processing in object-oriented databases. In: Proceedings of 10th International Conference on Data Engineering, pp. 518–525 (1994)