ISSN No. 0976-5697

Volume 1, No. 2, July-August 2010

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

CLUSTERING OBJECTS IN OBJECT-ORIENTED DATABASES BASED ON VECTOR SPACE

Truong Ngoc Chau
IT Faculty, Danang University of Technology,
Viet Nam
truongngocchau@yahoo.com

Abstract. Clustering objects in the storage is an important problem in Object-Oriented Database Systems. Based on the relationship sets of objects found in the system, we describe the objects as vectors in *m*-dimensional space, where *m* is the number of relationship sets found. After that, we define a distance between pairs of objects in order to build a clustering sequence of all existing objects in the system with their smallest total distance.

Keywords: object-oriented database, clustering objects.

I. INTRODUCTION

Clustering data in the storage is an important problem in designing physical database in generally. The way of storing related data in secondary storage units to improve performance of data queries is essential, so clustering can improve performance of database systems and it is considered as the principle problem which need solving database management systems.

In object-oriented database systems (OODBs), clustering means storing related objects to make them closer on secondary storage, so that when one object is accessed from disk, all its related objects are also loaded into memory. Then accessing to these related objects in a main memory access that is much faster than disk access. Many clustering algorithms be proposed, as Cactis [1], CK [2] and ORION [3]. Speciality, in OODBs, an object may be an instance of a class, and it may be an instance of a complex object too. Beside aggregation relationships and generalization relationships, other structural relationships such as version and configuration can also exist in parallel. In that case, objects have many close relationships, these influences clustering methods of objects. To solve these problems, authors in [5] proposed a solution in order to arrange objects' possible relationships by using distance matrix, then arrange these objects into a clustering sequence. However, the built in distance formula [5] is very complex, because we must calculate the probability of each relationship set that was clustered in the system and size of this relationship set.

In this paper, we continue studying types of objects' relationships in object-oriented database, then we quantize each object from a qualitative *m*-tuple to a numeric *m*-dimensional coordinate system similar a vector in *m*-dimensional space. On *m*-dimensional vector space, we build a Euclidian distance to calculate distance between any two objects in system. Based on the calculated distance, we create a distance matrix and arrange these objects into a clustering sequence.

The remaining sections of this paper are organized as follows: In Section 2 we present some solutions of clustering objects proposed, then we propose the idea of our objects cluster. Section 3 presents our proposed approach, including the Euclidian distance and distance matrix. Section 4 proposes the algorithm of arranging sequence of objects in the array elements with smallest total distance. Section 5 is conclusion.

II. RELATED WORKS

In order to execute data queries in the systems efficiently, normally, it depends on many factors, such as: the speed of CPU, memory, Input/Output disk costs, and communication resources. To optimize queries, in common, they specially focus on optimizing the semanticist of the query [6] and rarely mention optimizing I/O costs and other factors. In order to execute a query, the data is transferred from disk into main memory in term of block units, accessing data from a disk is normally much slower than accessing data from the main memory. Therefore, it is necessary to have a strategy to improve accessing data from

disk blocks for large databases which can't be stored in the main memory. One of the most efficient strategies is storing related information in continuous blocks; this strategy is also called data clustering.

The algorithm used most commonly is the K-Means algorithm [7]. This algorithm assigns the clusters based on factors provided by the user, such as the number of clusters, and the number of chosen center points. This algorithm is not suitable for the clustering of objects in the environment where exists in many relationships. Another approach was proposed in [8], the authors proposed the cluster techniques to minimize access time, in this approach the cluster object is based on the combination of probability and it can cluster objects in multiple relationships. Authors [1][2][3] gave many solutions to choose the strategy of clustering objects:

1. Clustering all objects belonging to the same class in the same segment of disk pages following *instance-of* member relationship.
2. Clustering all objects belonging to a class hierarchy rooted at a user specified class following *is-a* relationship.
3. Clustering all objects having the same inside reference according to *part-of* partial relationship. In that case, objects can belong to different classes.
4. Combining the 2nd and 3rd clustering techniques.

Each of clustering ways can optimize a query, but can't optimize other queries. For example, clustering of all objects following *part-of* relationship into the same block may cause objects from the same class could be stored on many pages instead of being stored in the same block. However, the manner of high-level relationships of objects having many relationships is a secular constraint in object-oriented systems. So it's necessary to have a method to process efficiently implementation clustering objects having high-level relationships in order to improve the performance of queries.

Authors in [4] and [5] proposed the way of using distance to measure relationship levels of objects. Each object is classified into relationships that it can take part in, presented by an *m*-tuple like that: $(category_1, \ldots, category_m)$, if there are *m* relationship sets in the system.

Example 1. There's an object database as the following:

```
define class TEACHER:
    type tuple( name: String;
                degree: String;
                department: DEPARTMENT;
                )
    end TEACHER

define class DEPARTMENT:
    type tuple( departmentName: String;
                dean: TEACHER;
                teacher: set(TEACHER);
                )
    end DEPARTMENT
```

Supposedly, we have objects shown as the following from the database above:

$O_1 = (i_1, atom, 'Tran\ Van\ Nam')$

$O_2 = (i_2, atom , 'PhD')$

$O_3 = (i_3, tuple, <name: i_1, degree: i_2, department: i_5>)$

$O_4 = (i_4, atom, 'Communication')$

$O_5 = (i_5, tuple, departmentName: i_4, dean: i_3, teacher: i_6)$

$O_6 = (i_6, set, \{i_3\})$

And the complex object instance of class *TEACHER* is denoted as CO_{tch}, *DEPARMENT* is denoted as CO_{dp}, the class which contains objects having atomic values is denoted as C. Hence, Object O_3 is a tuple of (*TEACHER*, CO_{dp}), we can consider this object as an instance of the class *TEACHER*, and belongs to the complex object CO_{dp}.

The problem is that almost all the values in every category of *m*-tuple are qualitative and discrete data. So, in order to measure distance between every two objects, we need to have methods to convert *m*-tuple into correspondent numerical data, so that, we can measure their distances. There are two basic constraints which are satisfied when we convert *m*-tuple into numerical data [4][5]:

1. Any calculation configuration has to ensure that objects which have closer relationships must have smaller distances.

2. If inside relationship manner of two pair of objects can't be compared distance after conversion, they shouldn't be compared for any reason.

We can describe the constraints above with following symbols:

θ: Semantic relationship between two objects, showing object's relationships.

α: The closer than operator.

- Constraint 1: *If* $\theta\ (O_1,O_2)\ \alpha\ \theta(O_1,O_3)$ *then* $d(O_1,O_2) < d(O_1,O_3)$
- Constraint 2*: If not(*θ*(O_1,O_2)* α *θ(O1,O3)) then not(d(O1,O2)* ρ *d(O1,O3))*, where ρ is comparison operator less or greater.

Example 2. From the database in Example 1, we have two objects O_1, O_2 from the same class *C*, and they both belong to CO_{tch}, but O_1 and O_4 only belong to the same class *C*. Therefore, $(O1,O2)\ \alpha\ (O1,O4)$, so we can infer that $d(O_1,O_2) < d(O_1,O_4)$ (*d* is the distance).

In order to show the measurement among the objects above, the authors in [5] have defined a metric measure to measure the distance between each pair of specific objects in any system. However, this measure is complex. Because each object is represented as *m*-tuple, we quantized each object from a qualitative *m*-tuple into a vector in *m*-dimensional coordinate system from the *m*-dimensional coordinate system, we can use euclidian distance measuring to measure distance between two objects. Our approach to ignore the 2nd constraint, we find that If inside relationship manner of two pair of objects can't be compared, then their distance is equivalent.

III. DISTANCE MEASURING BASED ON VECTOR SPACE

Relationship set is defined as a set of constrained objects in that relationship, and the size of that relationship set is the number of objects taking part in that relationship set. For example, class *DEPARTMENT* in Example 1 is considered as a relationship set, its size is the number of instances of this class. Another relationship set is the complex object instance, for example, CO_{tch} and its size is the number of objects in this complex object instance. We denote each relationship set as R_j and relationship sets existing in OODBs is $R = \{R_1, R_2, \ldots, R_m\}$, with *m* is the number of relationship sets.

Supposedly $\{O_1, O_2, \ldots, O_k\}$ is a set of all existing objects in a system. Each object is descried with respect as a set of *m* initial attributes in term of a vector.

$$\vec{O}_i = (O_{i1}, O_{i2}, ..., O_{im})$$

in *m*-dimensional space and

$$O_{ij} = \begin{cases} 0 & if \quad O_i \in R_i \\ 1 & if \quad O_i \notin R_i \end{cases}$$

with $1 \le i \le k$, $1 \le j \le m$.

Now we'll present an algorithm aiming to build vectors for each corresponsive object in the system.

Algorithm 1. Building vectors for each corresponsive object in the system.

Input.

- Set of objects in the system $W = \{O_1, O_2, \ldots, O_k\}$
- Set of relationship sets $R = \{R_1, R_2, \ldots, R_m\}$

Output. Set of vectors $V = \{\vec{O}_1, \vec{O}_2, ..., \vec{O}_k\}$ is described in *m*-dimensional space with

$$\vec{O}_i = (O_{i1}, O_{i2}, ..., O_{im}),\ 1 \le i \le k.$$

Method.

(1) V := {};
(2) For each O_i *in W do*
(3) Begin
(4) For j :=1 to m do
(5) If $O_i \in R_j$ *then* $O_{ij} := 0$*; else* $O_{ij} := 1$*;*
//construction of vector coordinates $\vec{O}_i$
(6) $V := V + \{\vec{O}_i\}$*;*
(7) End;
(8) Return V;

We can easily see the complexity of the algorithm dependent on loops (2) and (4), the loop (2) execute *k* times, the loop (4) execute *m* items, the complexity of the algorithm is *O(km)*.

Example 3. We assume that only two types of relationship existing in system are *instance-of* and *part-of*. Therefore, we have a set of relationship sets in Example 1 is {*TEACHER*, CO_{tch}, *DEPARTMENT*, CO_{dp}, *C*}, assume that set of relationship sets is ordered in a certain way (as the example above), then objects are described as vectors in 5-dimensional space correspondently with relationship sets described in Table 1.

Table I. Set of vectors corresponding objects

	TEACHER	CO_{tch}	***DEPARTMENT***	CO_{dp}	***C***
$\vec{O}_1$	***1***	***0***	***1***	***1***	***0***
$\vec{O}_2$	***1***	***0***	***1***	***1***	***0***
$\vec{O}_3$	***0***	***1***	***1***	***0***	***1***
$\vec{O}_4$	***1***	***1***	***1***	***0***	***0***
$\vec{O}_5$	***1***	***0***	***0***	***1***	***1***
$\vec{O}_6$	***1***	***1***	***1***	***0***	***1***

Because an object is described as a vector in *m*-dimensional space, so, metrics are used to measure the distance between any two objects. We use the Euclidian distance function to measure the distance between two objects O_i and O_j as the following:

$$d(O_i, O_j) = d(\vec{O}_i, \vec{O}_j) = \sqrt{\sum_{k=1}^{m} (O_{ik} - O_{jk})^2} \qquad (1)$$

For example, from distance formula (1), the distance between two objects O_1 and O_2 is the correspondent to the distance between two vectors

$$d(O_1, O_2) = d(\vec{O}_1, \vec{O}_2) = \sqrt{\sum_{k=1}^{5} (O_{1k} - O_{2k})^2} = 0$$

because O_1 and O_2 belong to the same relationship sets. Similarly, distance between $\vec{O}_1$ and $\vec{O}_4$ is

$$d(O_1, O_2) = d(\vec{O}_1, \vec{O}_2) = \sqrt{\sum_{k=1}^{5} (O_{1k} - O_{4k})^2} = \sqrt{2}$$

Distance matrix of k objects $\{O_1, \ldots, O_k\}$ is a matrix having the size *k x k*, and is defined like that:

$$M = (d(O_i, O_j))_{k \times k}.$$

Matrix *M* is symmetric by the principle diagonal and elements on the principle diagonal have the value of 0. So, in order to calculate matrix *M*, we only need to calculate elements *M(i, j)* with $j > i$.

IV. CLUSTERING ALGORITHM

Because we have to arrange all objects into the secondary storage, so the best way is arranging all objects of the system into a one-dimensional structural array, then mapping this one-dimensional structural array into the secondary storage. Therefore, our algorithm's outlet result is a sequence of all objects in the system, it satisfies that the sum of distances among objects in the sequence is minimum. With this algorithm, all properties of high-level relationships are still preserved. Our processing can be generalized in Figure 1.

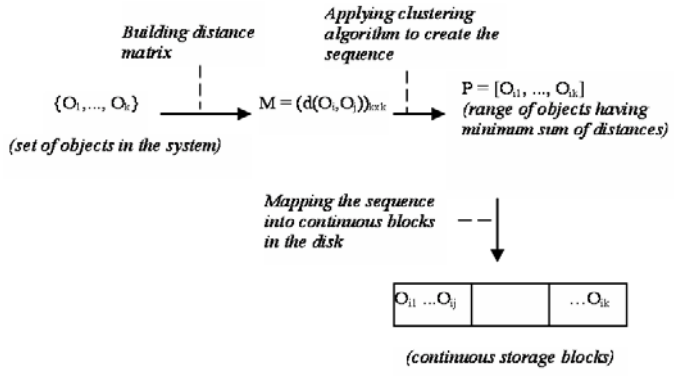

Figure 1. The processing to implement clustering objects.

After distance matrix $M = (d(O_i, O_j))_{k \times k}$ is built by measuring distances between each two objects based on applying Algorithm 1, clustering algorithm is used to arrange objects into a clustering sequence. The input of the algorithm is distance matrix *M* and $W = \{O_1, O_2, \ldots, O_k\}$ is a set of objects in the system, each object can take part in more than one relationship set. To be simple, we can consider *M* as weight matrix. From that, our algorithm will determine a line going through all vertices of a graph, each vertex for one time, this satisfied that the sum of distances between each two vertices (objects) in the sequence is minimum.

Algorithm 2. Create clustering sequence of all objects in the system has the smallest total distance.

Input.

- Distance matrix $M = (d(O_i, O_j))_{k \times k}$
- The set of all objects need to be clustered

$$W = \{O_1, O_2, \ldots, O_k\}.$$

Output. The clustering sequence including all objects $\{O_1, O_2, \ldots, O_k\}$, satisfying that the sum of distances between each two objects in the sequence is minimum.

Method.

(1) t := O_i; / choose a random object $O_i \in W$*/*

(2) h := 1;

(3) p[h] := t;

/ array p stores the clustering sequence of objects of W, satisfying that the sum of distances between each two objects in the sequence p is minimum}*/*

(4) W := W - { t };

*/*eliminate objects considered outside from W*/*

(5) While (h ≤ k) and (W <> ∅) do

Begin

(6) $d := \infty$;

(7) For each $O_j \in W$ do

(8) If ($d[t, O_j] < d$) and ($t <> j$) then

Begin

(9) $d := d[t, O_j]$;

(10) $p[h] := O_j$;

End;

(11) $t := p[h]$;

/* O_j is the chosen object for the next selective step*/

(12) $h := h + 1$;

(13) $W := W - \{t\}$;

End;

(14) Return p;

In order to consider the calculation complication of this algorithm, we see 2 loops (5) and (7) which are overlapped, each cycle implements maximum k times, so the number of times of implementing loop (7) is $O(k^2)$. As a result, the algorithm's complication is $O(k^2)$. The output result of the algorithm depends on choosing random object O_i from original W.

Example 4. Consider the object-oriented database given in Example 1, we have:

- $k = 6$: there are totally 6 objects,

- $m = 5$: {*TEACHER*, CO_{tch}, *DEPARTMENT*, CO_{dp}, C}, there are totally 5 set of relationship sets.

Then, the objects are described as vectors in the 5-dimensional space with 5 relation sets order (*TEACHER*, CO_{tch}, *DEPARTMENT*, CO_{dp}, C) as follows:

$\vec{O}_1$ = (1, 0, 1, 1, 0)

$\vec{O}_2$ = (1, 0, 1, 1, 0)

$\vec{O}_3$ = (0, 1, 1, 0, 1)

$\vec{O}_4$ = (1, 1, 1, 0, 0)

$\vec{O}_5$ = (1, 0, 0, 1, 1)

$\vec{O}_6$ = (1, 1, 1, 0, 1)

Step 1. Calculating the distance matrix

$M = (d(O_i\ O_j))_{6 \times 6}$

Because Matrix M is symmetric by the principle diagonal, we only have to calculate elements $d(O_i, O_j)$ with $j > i$. Distance matrix M is calculated based on given objects as the following:

$$M = \begin{pmatrix} & O_1 & O_2 & O_3 & O_4 & O_5 & O_6 \\ O_1 & 0 & 0 & 2 & \sqrt{2} & \sqrt{2} & \sqrt{3} \\ O_2 & & 0 & 2 & \sqrt{2} & \sqrt{2} & \sqrt{3} \\ O_3 & & & 0 & \sqrt{2} & 2 & 1 \\ O_4 & & & & 0 & 2 & 1 \\ O_5 & & & & & 0 & \sqrt{3} \\ O_6 & & & & & & 0 \end{pmatrix}$$

Step 2. Apply the clustering algorithm above, with the firstly chosen object is O_2, the sequence of objects with minimum sum of distances is $4+\sqrt{2}$ like that: O_2, O_1, O_4, O_6, O_3, O_5 (creating clustering sequence, we are installed on the *C++* programming language).

Because the secondary storage access unit is a disk segment or a block. Therefore, clustering sequence must be cut into segments according to ratio of object size to the block size (b/s). Notice that our clustering sequence is not interrupted by segmenting the clustering sequence. The change only occurs at the segmentation boundary. This can still be compensated by placing the adjacent clustering sequence segments into the contiguous neighboring blocks, which can save disk access time.

V. CONCLUSION

Based on relationships existing in object-oriented database system, we present objects in the system as vectors corresponding to the coordinate system which has a number of dimension equal to the number of relationship sets. From these vectors, we defined measurement the Euclidian distance between any two objects, and then we have the distance matrix. Based on distance matrix, we built an algorithm to create a sequence containing all objects in the system based on distance matrix. This ensures that sum the sum of distances among objects in the sequence is the minimum. With calculation complication of $O(k^2)$, in some cases, the algorithm may not give an optimal solution. However, with a large number of objects and objects take part in many different relationships, this algorithm is still applied efficiently. About mapping a sequence of objects into continuous blocks in the secondary storage, we will present in the next research result.

VI. REFERENCES

[1]. S.E.Hudson and R.King Cactis, "A Seft-Adaptive Concurrent Implementation of an Object-Oriented Database Management System", ACM Transactions on Database Systems, 14(3), September, 1989.

[2]. E.E. Chang and R.H. Katz, "Inheritance in computer-aided design databases: semantics and implementation issues", CAD, 22(8), October, 1990.

[3]. W. Kim and J.F. Garza and N. Ballou and D. Woelk, "Architecture of the ORION Next Generation Database Sytem", IEEE Transactions on Knowledge and Data Engineering, 2(1), March 1990.

[4]. Steven Yi-Cheng Tu and Daniel J.Bueher, "Clustering Objects for OODBS in a Multiple Relationship Environment", Pan Pacific Conference on Information Systemss, 1993.

[5]. D.V. Ban and T. N. Chau, "Cluster Objects in Object-Oriented Database by Using Distance Matrix", Journal of computer science and cybernetics, VietNam, 25(2), pp.178-187, 2009.

[6]. Trigoni and Agathoniki, "Semantic Optimization of OQL Queries", University of Cambridge, Computer Laboratory, UCAM-CL-TR-547, ISSN 1476-2986, October, 2002.

[7]. LingWang and Liefeng Bo and Licheng Jiao, "A modified K-Means clustering with Density-Sensitive Distance Metric", Institute of Intelligent Information processing, Xidian University Xi'an 710071, China, 2008.

[8]. Vlad S. I. Wietrzyk and Mehmet A. Orgun, "Clustering techniques for minimizing object access time", Springer, Vol 1475, pp.236-247, 2004.