A Partitioning strategy for OODB

Dr. Sudesh Rani
Asstt. Prof.(Computer Science), Govt. College, Hisar
Kurukshetra University, Kurukshetra, Haryana, India

Abstract

An effective strategy for distributing data across multiple disks is crucial to achieving good performance in a parallel object-oriented database management system. During query processing, a large amount of data need to be processed and transferred among the processing nodes in the system. A good data placement strategy should be able to reduce the communication overheads, and, at the same time, to provide the opportunity for exploiting different types of parallelism in query processing, such as intra-operator parallelism, inter-operator parallelism, and inter-query parallelism. However, there exists a conflict between these two requirements. While minimizing interprocessor communication favors the assignment of the whole database to a small number of processors, achieving higher degree of parallelism favors the distributions of the database evenly among a large number of processors. A trade-off must be made to obtain a good policy for mapping the database to the processors.We need good heuristics to solve this and more complicated database allocation problems. In this paper, we propose some heuristics for partitioning an OODB so that the overall execution time can be reduced.

Keywords: Parallelism, Vertical partitioning, Horizontal partitioning, Query diameter.

1. Introduction

In order to achieve parallelism, the database needs to be partitioned over multiple components in a parallel system. For example, relations in Gamma ([4], [5]) are horizontally partitioned across all nodes with disk drives using one of four declustering strategies provided in the system: round-robin, hashed, range, and hybrid-range partitioned. However, none of the strategies is a clear winner in the performance analysis ([7], [8]). To decluster all relations across all nodes with disks is recognized as a serious mistake ([5]). A better solution used in Bubba ([3]) is to decluster a relation based on the \heat" (i.e., the cumulative access frequency) and the size of the relation. Since the ideal data placement changes continuously as the workload changes in time, Bubba repeatedly refines the data placement if the performance improvement is worth the work required to reorganize.

In a relational database environment, a relation may be accessed by several types of queries which require different sets of attributes. In order to improve the performance, attributes of the relation are divided into groups and the relation is projected into fragment relations according to these attribute groups. This process is called vertical partitioning. The fragments are assigned to different sites in distributed database systems to minimize the cost of accessing data by all queries.

There are trade-offs between horizontal and vertical partitioning methods. A general discussion of pros and cons on a decomposed storage system (DSM), which pairs each attribute value with the surrogate of its record, is reported by Copeland and Khoshafian ([2]). Several parallel database projects have employed some form of the same vertical data partitioning concept ([9], [10], [11], [14]). A simple file assignment problem, which deals with assigning files to different nodes of a computer network, has been studied extensively ([6]). However, most of the works assume that a request is made at one site and all the data for answering it is transferred to that site. This simple view of application cannot model the query processing strategy in a parallel database system. The simple file assignment problem is an NP-complete problem

As for the OODBs, it is recognized that object clustering is important to the performance ([1], [12], [13]). However, the clustering in OODBs is still an open research issue, and therefore the problem of declustering an OODB for a parallel system is a new challenge in research.

2. The Problem

The problem is to partition a given OODB and assign the partitions to the nodes in a multiprocessor system. It is assumed that the number of object classes in the database is larger than the number of processors in the system. Also, we assume that the processors are fully connected. This simplifies the problem so that we do not need to consider the effects of the network's physical topology. However, we can simulate different topologies by introducing various delays to different links.

We assume that the unit of distribution is class. In other words, classes are not allowed to be split, and each class must reside in one and only one node. Since we group all the data associated with an object class together, we can localize retrieval, manipulation, and user-defined operations and reduce the overall communication among processors. If we horizontally partition the classes and assign them to multiple processors, two sets of processors need to communicate with each other when two classes want to exchange information. In addition, if a large number of processors work on the same class, this horizontal partition scheme does not provide a good environment for multiple queries to be executed in parallel when these queries access different classes. Thus, we choose class as the unit of partition in this study. However, if some classes are too large for one node to handle and we decide to split them, the heuristics presented in this paper can still be used to group the partial classes.

It is not easy to find a partition which is good for all the applications. A good partition for one application may not be suitable for another application. If we make a compromise for both applications, neither one will perform well. Therefore, we decide to partition the database based on the processing requirement of a single application which is characterized by a set of typical queries used in the application. By analyzing the query patterns in the set and the data characteristics of the database, we try to find a partition so that the execution time of the set of queries is minimized.

If we want to calculate the execution time of a query, an appropriate cost function is needed for modeling the parallel execution of the query. For a set of queries, the interaction and interference among queries will make it extremely difficult to formulate the cost function. Even if we can formulate the correct function, the problem of finding the minimum would be intractable. Therefore, instead of finding the best partition which gives the minimal execution time, we try to find some heuristic rules that will avoid bad partitions and give good performance.

3. Heuristics for Partitioning an OODB

The execution time of a query in a parallel environment consists of three components: CPU time, IO time, and communication time. Since the CPU time and IO time in each processor are the time the processor works on the query, we use the term "processing cost" to represent these two time components. It takes some communication time for a message to transfer from one processor to another. However, both the source and the destination processors can do other tasks during this time.

If the communication delay is short, one obvious bad solution for partitioning the database is to assign all object classes to one single node. In other words, we want to balance the processing load on the nodes as well as to reduce the communication cost among them. However, we cannot use the sum of the processing cost and the communication cost as the total cost for a partition because it is difficult to give a meaning to the combined cost. Also, if we use the combined cost to partition the database, we run the risk of having two equal cost partitions in which one has high processing cost and the other one has high communication cost. On the other hand, **if the communication delay is long,** we want to group classes that exchange large amount of data and reduce the length of the "path" through which messages and data must be transferred. Therefore, we try to find a combined heuristics for partitioning the database.
The heuristic method is based on the overall processing cost of each class referenced in a query. We measure the overall CPU time and IO time used for processing a class to represent the processing cost of that class in the query. When we consider the set of queries, we take the sum of the processing costs for the same class in all queries to represent the total processing cost for that class.

Figure 1 shows the example university database with a class number and a class size in parentheses (i.e., the number of instances) attached to each class. A set of 10 queries as shown in Figure 2 represents the processing requirement of a specific application that we want to partition the database for. In this example, we have 5 simple queries (queries 0, 1, 2, 3, and 4) and 5 complex queries (queries 5, 6, 7, 8, and 9). Each simple query contains 3 or 4 classes and each complex query has 6 or 7 classes. Since this set of queries has a large variety of query patterns, we feel that it can represent a general application of this database. The number in parentheses beside each class number is the measured processing cost of the class when the query is actually executed. We can calculate the processing cost of each class. For example, class 2 (Transcript) has been referenced twice in query 0 and query 8. The overall processing cost of class 2 in the set is the sum of the processing costs of class 2 in query 0 and query 8. Therefore, the overall processing cost of class 2 is 46.13. The calculated processing costs for all the classes referenced by the query set are shown in Table 1**.**

The overall processing cost of a class represents the minimal work that needs to be done for the set of queries if the class is assigned to a single processor. If we assign multiple classes to a processor, the load of the processor is the sum of the processing costs of the classes that are assigned to it. In order to achieve good performance, we want to distribute the load among the processors as evenly as possible. Load balancing is our main consideration for partitioning the database.

However, when we group two classes and put them on the same processor, the time for exchanging messages between these two classes can be drastically reduced. Therefore, we also want to group classes in a way so that the overall communication time can be reduced. The length of the longest path in a query is called the *query diameter***.**

If we reduce the diameter of a query, the overall communication time of the query will also be reduced. The query diameter can be reduced by grouping adjacent classes in the longest path and assigning them to the same processor. This is our secondary consideration for partitioning the database.

We combine the above two heuristics into the following method for partitioning a database. Since we want to evenly distribute the processing cost among the processors, the number of groups of classes that we formed should be equal to the number of processors, assuming the number of classes is larger than the number of processors.

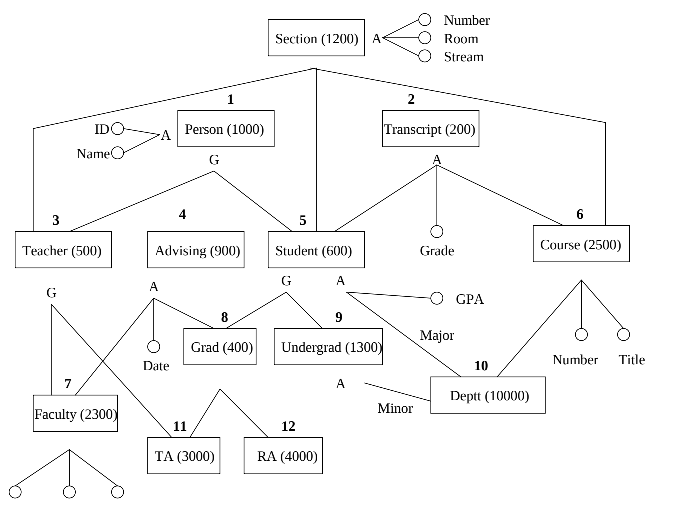


Fig1: The University database

The overall processing cost of each group should be close to the average processing cost among the processors. The average processing cost is called threshold cost. If some of the classes have processing cost that are larger than the average processing cost of the processors, we assign each of them to an empty group and will not assign any other class to these groups. The remaining classes should be distributed among the remaining processors as evenly as possible. Since the processing costs of the classes assigned to the single-class groups are above the threshold cost, the average processing costs of the remaining classes would be lower than the threshold cost. For this reason, we calculate a new threshold cost based on the costs of the remaining classes.

This new threshold cost is used as an upper limit for grouping classes in the first phase. When we group classes together, the total processing cost of the resulting group should not be larger than the new threshold cost. We start from the query with the largest diameter in the set and try to reduce the diameter by grouping two adjacent classes in the longest path. The two adjacent classes with the smallest combined processing cost will be considered. If the combined cost does not exceed the threshold cost, we group them together and use the combined cost as the processing cost of these two classes in all the queries. This step reduces the length of the longest path by 1. Then, we try to reduce the next longest path in the set by 1.

If there are multiple paths with the same length, we find a candidate pair of class for each path and choose the pair with the lowest combined cost to group. This process will continue until we cannot reduce the length of any path by grouping classes or the number of groups is equal to the number of the processors. In this phase, while we group classes to reduce the query diameters, the threshold cost is used to control the load in each group so that we can balance the load during the second phase of our heuristic method.

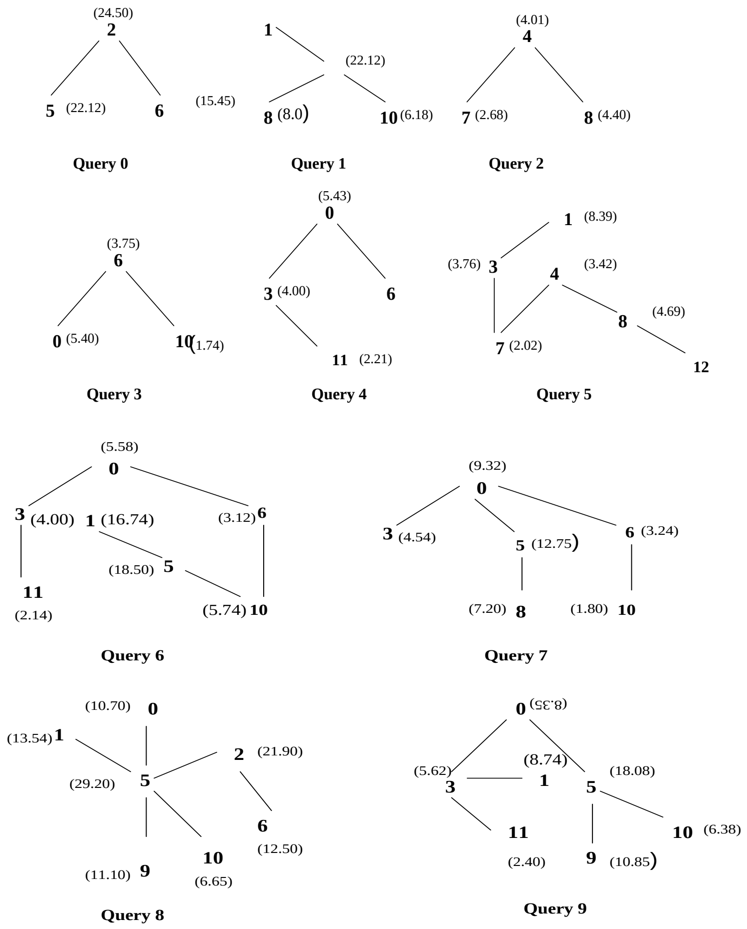


Fig 2: Sample Queries

Class No.	Cost
0	44.78
1	60.26
2	46.40
3	21.92
4	7.43
5	118.68
6	42.34
7	4.70
8	24.29
9	21.95
10	28.49
11	6.75
12	3.59

Table 1: Processing cost of each class

After we finish grouping classes for reducing query diameters, we need to reduce the number of groups to the number of processors in the system. In other words, we want to form the same number of clusters of groups as the number of processors. First, the groups are sorted based on their processing costs. Then we assign the group with the largest cost to the first available cluster with the lowest cost and add the group's cost to the cluster's cost. By continuing this simple process, we can assign all groups to a fixed number of clusters having relatively close final costs among the clusters. Then, we can assign each cluster to a processor because we assume all the processors are the same and they are fully connected.

4. An Example

If we want to partition the university database for a 7-node system, we need to find a suitable set of queries to represent the application and measure the processing cost of each class in each query. An example is shown in Figure 2 Then, we calculate the overall processing cost of each class and the results are shown in Table 1. The next step is to find the threshold cost. Since the total processing cost of all classes is 434.59 and the average processing cost of the 7 processors is 62.08, the cost of class 5 is too large for it to be considered in the following procedure. Therefore, we just assign class 5 to a processor and drop it from further consideration. We also re-calculate the average cost of the remaining 6 processors and it is 52.65. This is the new threshold cost.

Among the 10 queries, query 6 has the longest diameter of 6. The adjacent classes 3 and 11 have the smallest combined cost (29.06) in the longest path in the query. We group them together. Now, queries 6 and 5 both have a diameter of 5. We check the longest path in query 6 and cannot find two adjacent classes that have a combined cost lower than the threshold cost. In query 5, we find classes 4 and 7 can be grouped together. By continuing this process, we find that the groups with their cost in parentheses are as follows: 5 (118.72); 1 (60.90); 9 and 10 (50.83); 2 (46.13); 0 (45.26); 6 (43.01); 4, 7, 8, and 12 (40.68); 3 and 11 (29.06).

We assign the 7 largest groups to the 7 empty clusters. Then, we assign the next group (in this case 3 and 11) to the lowest cluster. After we finish all the assignment, we have the following clusters: class 5; classes 3, 11, 4, 7, 8, and 12; class 1; classes 9 and 10; class 2; class 0; class 6. The heuristics presented here along with some other methods will be evaluated in the following chapter.

5. Evaluating Partition Heuristics

This method performs better than LB and OCPN methods of partitioning. The LB heuristic method does not try to reduce the query diameters in the query set. It directly goes to the second step and tries to balance the load. The one-class-per-node partitioning method (OCPN) assigns only one class to a node. Partition of our method performs better than the LB and OCPN methods when the communication delay increases. This means the heuristics used for partition the database is a good one.

6. Conclusion

We have proposed a heuristic method for partitioning the database. The database is partitioned for a specific application the processing requirement of which is represented by a set of queries. By analyzing the queries and the system characteristics, we can partition the database to suit the application. This heuristic method first uses a threshold cost as a guide to group small classes so that the query diameters can be reduced. Then, it tries to evenly distribute the cost among all the processors. This heuristic method is based on the overall processing cost of each class referenced in a query. We measure the overall CPU time and IO time used for processing a class to represent the processing cost of that class in the query. When we consider the set of queries, we take the sum of the processing costs for the same class in all queries to represent the total processing cost for that class.This method performs better than other partitioning methods e.g. LB and OCPN if the communication delay is long.

7. References

[1] J. R. Cheng, and A. R. Hurson, “Effective clustering of complex objects in object-oriented databases”, in ACM SIGMOD International Conference on Management of Data, Denver, CO, 1991, pp. 22-31.

[2] G. Copeland, and S. Khoshafian, “A decomposition storage model”, in ACM SIGMOD International Conference on Management of Data, Austin, TX, 1985, pp. 268-279.

[3] G. Copeland, W. Alexander, E. Boughter and T. Keller, ”Data placement in Bubba”, in ACM SIGMOD International Conference on Management of Data, Chicago, IL, 1988, pp. 99-108.

[4] D. J. DeWitt, R. Gerber, G. Graefe, M. Heytens, K. Kumar and M. Muralikrishna, ”GAMMA-A high performance dataflow database machine”, in 12th International Conference on Very Large Data Bases, Kyoto, Japan, 1986, pp. 228-237.

[5] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H. I. Hsiao and R. Rasmussen, “The Gamma database machine project”, IEEE Transactions on Knowledge Data Engg., Vol. 2, No. 1, 1990, pp. 44-62.

[6] L. W. Dowdy, and D. V. Foster, “ Comparative models of the file assignment problem” , ACM Computing Survey, Vol. 14, No. 2, 1992, pp. 287-313.

[7] S. Ghandeharizadeh, and D. J. DeWitt, “ Hybrid-range partitioning strategy: A new declustering strategy for multiprocessor database machines”, in 16th International Conference on Very Large Data Bases, Brisbane, Australia, 1990, pp. 481-492.

[8] G. Ghandeharizadeh, and D. J. DeWitt, “A multiuser performance analysis of alternative declustering

strategies", in 6th International Conference on Data Eng., Los Angeles, CA, 1990, pp. 466-475.

[9] S. Khoshafian, G. Copeland, T. Jagodits, H. Boral and P. Valduriez. "A query processing strategy for the decomposed storage model", in 3rd International Conference on Data Engg., Los Angeles, CA, 1987, pp. 636-643.

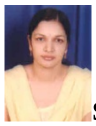
[10] S. Khoshafian, P. Valduriez and G. Copeland, " Parallel query processing for complex objects", in 4th International Conference on Data Engg., Los Angeles, CA, 1988, pp. 202-209.

[11] H. Lam, S. Y. W. Su, F. L. C. Seeger, C. Lee and W. R. Eisenstadt, "A special function unit for database operations within a data-control system", in International Conference on Parallel Processing, Chicago, IL, 1987, pp. 330-339.

[12] K. Shannon, and R. Snodgrass, "Implementing Persistent Object Bases: Principles and Practice", Morgan Kaufmann Publishers, Palo Alto, CA., 1991, pp. 389-402.

[13] M. M. Tsangaris, and J. F. Naughton, "A stochastic approach for clustering in object bases", in ACM SIGMOD International Conference on Management of Data, Denver, CO, 1991, pp. 12-21.

[14] P. Valduriez, "ACM Transactions on Database System", Vol. 12, No. 2, 1987, pp. 218-246.



Sudesh Rani got his Ph.D. degree in Computer Science from the Kurukshetra University, Kurukshetra, India, in 2009, on "Algebraic query processing and parallelism in databases". Her areas of interest are data mining, parallel databases, query processing in databases etc. Presently she is working as Asstt. Professor in Computer Science at Govt. College, Hisar, Kurukshetra University, Kurukshetra, India