A New Multidimensional Data-Space Placement Algorithm*

TAN Yu-song[1], YANG Li[2], ZHOU Xing-ming[1]

[1](National Laboratory for Parallel and Distributed Processing, National University of Defence Technology, Changsha 410073, China);

[2](Computer Software Center, NEU-Alpine Company, Shenyang 110006, China)

E-mail: ystan@nudt.edu.cn; yangli@neu-alpine.com; xmzhou@nudt.edu.cn

Received June 22, 1999; accepted July 12, 2000

Abstract: This paper describes a new multidimensional data-space placement algorithm, SMDPA. The algorithm can place the data hyper-cubes efficiently even they have different accessed frequencies. The algorithm uses the data hyper-cubes' prior accessed frequency and their similarities to get the accessed relation between them. The simulation results make clear that the SMDPA algorithm has better performance than traditional algorithms.

Key words: multidimensional data-space; similarity; parallel database

With the development of the database system's application, the database system has four new characteristics: very large storage, complex and unstructured data objects, complex data query and real-time requirement. Such systems arise in many applications, including CAD, GIS etc.

To match these characteristics the database system should develop an effective algorithm to partition the data-space and place the data hyper-cubes. There are a lot of similar algorithms, such as Disk Module[1], CMD[2], Field-wise Exclusive-OR Method[3], Error Correcting Coce Method[4], Hilbert Curve Method[5]. These algorithms are all multidimensional algorithms, which can place data hyper-cubes. Traditional algorithms are based on a hypothesis, i.e. all data pointers in the data space are accessed with equal frequency. In fact the actual application does not match this hypothesis, i.e. the data pointers may have different access frequenies. So these traditional algorithms just have theoretical significance and would not be effective in the practical system.

This paper studies a new data hyper-cube algorithm, Similarity-based Multidimensional Data-space Partition Algorithm, SMDPA for short, which is not based on that hypothesis and it assumes that all data pointers should be accessed with unequal frequency. So the SMDPA can be effective in real system. And since the SMDPA is a multidimensional data space partition algorithm too, it can meet the database system's new requirements.

The outline of this paper is as follows. In Section 1 we introduce some background theory about the problem. In Section 2 the paper gives the SMDPA algorithm. And it will give some simulation results of CMD/HCA/XOR/SMDPA algorithms in Section 3, while the results indicate that the SDMPA algorithm is very effective in actual database system.

* This project is supported by the National Natural Science Foundation of China under Grant No. 69673046 (国家自然科学基金). **TAN Yu-song** was born in 1976. He is a Ph.D. candidate of the National University of Defence Technology. His research areas include parallel database, CSCW and wireless network. **YANG Li** was born in 1962. He got his Ph.D. degree in 1995 at the National University of Defence Technology. Now he is a professor and CTO of NEU-Alpine Company. His research areas include database. **ZHOU Xing-ming** was born in 1938. He is a professor and doctoral supervisor of the National University of Defence Technology. He is a member of the Chinese Academy of Sciences. His research interests are computer architecture, database, CSCW etc.

© 中国科学院软件研究所 http://www.jos.org.cn

1 Background

First we define some terms that are used throughout this paper. These definitions are similar to those used by Ref. [6].

Definition 1 (Response Time). The response time of a query is defined as $\max(N_1,N_2,\dots,N_D)$, where $N_i (i \in [1,2,\dots,D])$ is the number of qualifying pages on ith PU for the query.

Within this definition we assume that just I/O is the bottleneck. CPU and network are so fast that we can ignore their cost.

Data-space Declustering Problem (DDP) can be described as follows. Given a D-dimensional Cartesian product data space, $D_1 \times \dots \times D_D$, in which D is the number of the data space's dimensions, and D_i denotes the domain range of the ith attribute. In the parallel database system there are N processing units (PU). How to decluster the data-space among the N PUs so that the maximum performance is achieved for data range queries is crucial. Unfortunately Ref. [7] proved that the DDP is an NPC problem generally. So we just can find an effective rather than the best algorithm to solve the DDP.

Since I/O is the major bottleneck in data query processing, it is desirable that I/O is parallel as far as possible. We think that in DDP the goal of achieving the maximum performance is to maximize the throughput and minimize the response time of the data queries.

We consider that the integrated algorithm for DDP should have two important parts. One is how to stripe the multidimensional data-space to some data hyper-cubes, called Data-space Striping Algorithm (DSA). And the other part is how to place these data hyper cubes among the N PUs, called Data hyper-cube Placement Algorithm (DPA). This paper just considers DPA and another paper will discuss DSA.

Most of traditional algorithms are based on one hypothesis, i.e. all data hyper-cubes are accessed by query with the same frequency. It is evident that the actual application does not match this hypothesis, i.e. the data records may have different access frequencies. So these traditional algorithms cannot be effective in actual system.

How to place the data hyper-cubes with different accessed frequencies? To answer this question, we use accessed frequency of all data pointers and the application's locality.

Every system/application has a natural characteristic that is locality. That is to say that the application tends to access some relative data frequently. Considering the locality we can place the data hyper-cubes that are accessed together frequently at different PUs to achieve high parallel performance. And we should place the hot hyper-cubes at cool PUs to achieve good balance performance. These are the core idea of our SMDPA algorithm.

There are some definitions about the data hyper-cube's accessed frequency as follows.

Definition 2 (Similarity). Considering the locality of data query operation, if hyper-cube A will be accessed with frequency p when hyper-cube B was accessed early, it is called that there is similarity (its value is p) between hyper-cubes A and B. Or we say that hyper-cube A is similar to hyper-cube B with p. Use $\mathrm{Sim}(A,B)$ to denote the similarity.

For some reason the system cannot log the data hyper-cubes' accessed order for data query, so this paper supposes that $\mathrm{Sim}(A,B)=\mathrm{Sim}(B,A)$.

Definition 3 (Prior Accessed Frequency). The parallel database system logs the data hyper-cubes' accessed frequency. Since these frequencies do not consider the similarity, we call this accessed frequency as prior accessed frequency. $PAF(A)$ denotes the prior accessed frequency of data hyper cube A.

Definition 4 (Posterior Accessed Frequency). Considering the prior accessed frequency of data hyper-cube and the similarity between hyper-cubes, we can calculate the posterior accessed frequency of every data hyper-cube, that is the actual accessed frequency that the system accesses the data hyper-cube. $PeAF(A)$ denotes the posterior

© 中国科学院软件研究所 http://www.jos.org.cn

accessed frequency of data hyper-cube A.

In another view the prior accessed frequency is the frequency that the parallel database system accesses the single data hyper-cube. The posterior accessed frequency is the frequency that the system accesses more than one hyper-cube, since now there are smiliarities between all accessed hyper-cubes. And the similarity just is the conditional frequency.

2 SMDPA Algorithm

In this section we will introduce SMDPA algorithm in detail.

How can we get the information of data accessed to process SMDPA? There are three methods to get the information. The first method is to suppose the system matches a given probability model and we can use that formula to get the information of accessing. The second method thinks that the parallel database system can get the prior accessed frequencies of all data hyper-cubes and the similarity between hyper-cubes by logs. The last and most complex method is using clustering algorithm to partition all data records into some clusters and supposes that all data records have the same accessed probability. The three methods can estimate the prior accessed frequency of every data record. And with the idea of locality we think that those data record pairs, which have large distance, have little similarity. Based on this information we can calculate posterior accessed frequency easily. It is true that these methods have different costs and different estimate precisions. So we can apply SMDPA to initial placing stage. After system runs and collects the accurate information of data accessing, SMDPA can be used to reorganize the data-space accuratly.

The core technologies of the algorithm are to get the posterior accessed frequencies of all data hyper-cubes and create some ordered linked lists.

Since the parallel database system accesses the data hyper-cubes with their posterior accessed frequency, we must get all data hyper-cubes' posterior accessed frequencies firstly. But since the system can access them with prior accessed frequency and there are similarities between hyper-cubes too, how the system accesses the hyper-cubes?

If we consider this question with the view of graph, it can be changed to a graph question as follows. Given a complete directional weighted graph $\langle V, E\rangle$, V is the set of points and each point has a weight which denotes the prior accessed frequency. E is the set of directional edges. There are two directional edges between two points and each edge has a weight too. The weight is the similarity between two points. How to partition the graph into some sub-graphs to get the maximum sum of all points?

It is sure that we must increase the points' weight using edges' weight as far as possible. So suppose we have gotten a point from the set of nonprocessing points, for example it is DC_i, which has the maximum weight of the set. We should delete all edges that the other points of the set point to it for it is accessed before others and the similarity between others and it does not exist. And we should use this point to decide whether we need to modify the other points' weight. If the current weight of one point, for instance it is DC_j, is smaller than the posterior accessed frequencies calculated, it will be modified. It is said that the DC_j will be accessed with high probability if DC_i has been accessed. So we should log the point with a ListID, which is the ID of the linked list that includes DC_i, for it has similarity to those points in that linked list. If one point's current weight is higher than the calculated result, it need not be modified. It is indicated that this point may be accessed separately with higher probability. And we will delete the edge that DC_i point to this point.

After we cluster the graph, we can get an uncompleted directional graph now, and get some separable directional sub-graphs. A sub-graph indicates that all points that are in this sub-graph will be accessed together with the highest frequency. The linked list created is ordered. The order denotes the sequence with which data hyper-cubes are accessed at posterior accessed frequency with descending order. To get good performance for data query we must place these points into different PUs. We use disk module method to place the points that are in one sub-graph with a certain start PU. As a result the system will have good parallel performance for data query.

To achieve good balance performance we should place the data hyper-cubes with high posterior accessed frequency to different PUs. Since the hyper-cubes in one linked list have a descending order, that is to say that we should place the different lists from different start PUs. The algorithm selects the PU which has the lowest accessed frequency to be the start PU ID of new linked list. Then the system will have good balance performance for data query.

When the system accesses the data hypercube, it should log the information. To reduce the system cost we can recalculate the values of accessed frequencies using interval as unit. And with the idea of locality, system will access some data records, which are in the same data hypercube with large probability. So though system accesses a great lot of data records, it just needs to recalculate a few accessed values of subintervals.

Now we give the SMDPA algorithm as follows:

```
Input:  {$PAF(DC_i)|i\in[1,2,...,K]$}, in which $K$ is the number of data hyper-cubes;
        {$\mathrm{Sim}(DC_i,DC_j)|i,j\in[1,2,...,K]$};
        $N$, which is the number of PUs.
Output: {$PUId(DC_i)|i\in[1,2,...,K]\wedge PUId(DC_i)\in[1,2,...,N]$}, it denotes the PU's ID, in which the
        DC is placed.
        /* Now initialize some environment variants about the algorithm. */
Step 1.  $DC:=\{DC_i|i\in[1,2,...,K]\}$
Step 2.  Prob($DC_i$):=$PAF(DC_i)$, $\forall i\in[1,2,...,K]$
Step 3.  ListId ($DC_i$):=NULL, $\forall i\in[1,2,...,K]$
Step 4.  NumOfLinkedList:=$-1$
Step 5.  WorkLoad($PU_i$)=0, $\forall i\in[1,2,...,N]$
        /* Now create some linked lists, which include all data hyper-cubes. The hyper-cubes in the same
           list will be accessed together. */
Step 6.  while ($\|DC\|>0$) do {
Step 7.    Find the $DC$ that has the maximum value of Prob. Assume it is $DC_i$
Step 8.    $DC:=DC-\{DC_i\}$
Step 9.    if ListId ($DC_i$)= NULL then {
Step 10.     NumOfLinkedList:=NumOfLinkedList+1
Step 11.     LinkedList (NumOfLinkedList)={$DC_i$}
Step 12.     ListId ($DC_i$)=NumOfLinkedList}
           else
Step 13.   LinkedList (NumOfLinkedList)=LinkedList (NumOfLinkedList)+{$DC_i$}
        /* Correct the posterior accessed frequency of other data hyper-cubes except $DC_i$. */
Step 14.   for (all $DC_j$ in $DC\wedge j\neq i$) do {
Step 15.     TempFrequency:=Prob($DC_i$)*Sim($DC_j$,$DC_i$)
Step 16.     if (TempFrequency>Prob($DC_j$)) then do {
Step 17.       Prob($DC_j$)=TempFrequency
Step 18.       ListId($DC_j$)=ListId($DC_i$)
             } /* End of if */
           } /* End of for */
```

```
            } /* End of while */
          /* Now place all data hyper-cubes among all PUs. */
Step 19. for ($i := 0$; $i <$ NumOfLinkedList; $i = i + 1$) do {
Step 20.     List = LinkedList($i$)
          /* Get the ID of the lowest accessed frequency to be the first ID to place the linked lists. */
Step 21.     $k$: = GetIdleNodeID();
Step 22.     visit all data hyper-cubes in List orderly, do {
          /* Assume visit the $DC_j$ */
Step 23.       PUId($DC_j$): = ($j + k$) mod $N$
Step 24.       WorkLoad(PUId($DC_j$)): = WorkLoad(PUId($DC_j$)) + Prob($DC_j$)
            } /* End of visits to all data hyper-cubes in one linked list */
          } /* End of visits to all linked lists to place the data hyper cubes */
```

3 Simulation Result

In this section we will give the simulation results of SMDPA algcrithm comparing with CMD/HCA/XOR algorithms. The results will prove that SMDPA has a better performance than traditional algorithms.

Now we give some definitions to define the performance of algorithms as follows:

Definition 5 (Degree of Parallel). Given a DPA algorithm, the data hyper cubes that satisfy a certain data query Q must be stored among some PUs. So we call the number of these store PUs as the degree of parallel for the query Q. It is denoted by $DP(Q)$.

Definition 6 (Degree of Balance). Given a DPA algorithm, for a certain query Q there are $DP(Q)$ PUs to be started. In these PUs there are some data hyper-cubes that satisfy Q, so we define the following formula as degree of balance for Q. It is denoted by $DB(Q)$.

$$\frac{\mathrm{Max}(\mathrm{NumOfCube}(\mathrm{PU}_i) \in \mathrm{PUSet}(Q)) - \mathrm{Min}(\mathrm{NumOfCube}(\mathrm{PU}_j) \mid \mathrm{PU}i \in \mathrm{PUSet}(Q))}{\mathrm{Max}(\mathrm{NumOfCube}(\mathrm{PU}_i) \mid \mathrm{PU}_i \in \mathrm{PUSet}(Q))}$$

in which, NumOfCube(PU_i) is the number of data hyper-cubes that satisfy the query and are stored in PU_i. And PUSet(Q) is the PU set which includes and only includes the PUs strated by query Q.

It is clear that a good algorithm should achieve high DP and low DB.

During simulation we first generate all data hyper-cubes with a certain frequency distribution. Then we use four algorithms as DPA to place hyper-cubes among the PUs. They are the CMD algorithm, the Hilbert Curve algorithm (HCA), the Field XOR algorithm (XOR) and the SDMPA algorithm. We generate some data range queries that match the frequency distribution, and use these queries tc test the algorithms' performance.

For all data hyper-cubes' frequencies created randomly, we assume that there is a peak point, near which the data hyper-cubes have higher frequency to be accessed than other area. So when generate the hyper-cube's PAF, we calculate the distance between the hyper-cube and the peak point first. The PAF has inverse ratio to the distance. And when generate the data query, it is intended that the queries access the area near the peak point.

And for SMDPA algorithm we should generate the similarity value between two data hyper-cubes randomly too. Considering the locality of the application, we think that the similarity value has relation, it is of inverse ratio too, with the distance between two hyper-cubes.

We will change three simulation's environment variants, which are the number of PUs, the number of data hyper-cubes and the length of data queries, to test the algorithms' performance.

The simulation results are as follows.

Figure 1 represents the total accessed frequency, that is the sum of all data hyper-cubes' accessed frequencies of every PU. It is clear that in four algorithms there is just one algorithm, that is SMDPA algorithm, which can maintain that all PUs have approximately equal total accessed frequency. And other three algorithms cannot do that. With those algorithms there are some PUs that are so hot that they will be the bottleneck of the system. But at the same time, there are some PUs that have so low access frequency that they will be idle at most time. For example, with Hilbert Curve algorithm the total accessed frequency of the third PU, whose ID is 2, is 0.13439, while the 12th PU's, whose ID is 11, is 0.0342095. The ratio of two values is as high as 4.

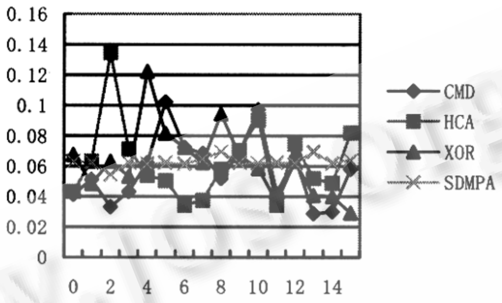

Fig. 1 The total accessed frequency of PUs (16/2500)

Figures 2 and 3 are the parallel and balance performance of four algorithms with 16 PUs and 900 data hyper-cubes. Figures 4 and 5 are the parallel and balance performance of four algorithms with 16 PUs and 2500 data hyper-cubes. And Figs. 6 and 7 are the parallel and balance performance of four algorithms with 32 PUs and 2500 data hyper-cubes. The abscissa of these six figures is the length of data queries, i. e. the length of one attribute of range query. And the ordinate of these figures is the parallel performance of algorithms (DP(Q)) or the balance performance (LB(Q)).

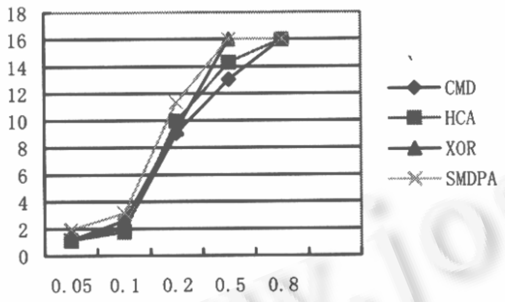

Fig. 2 The parallel performance of 16/900

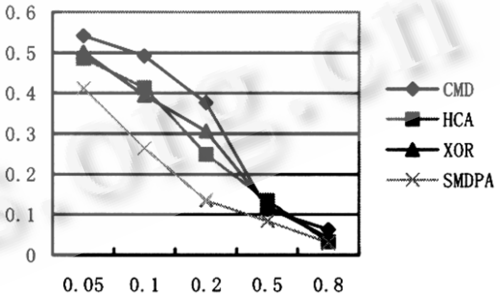

Fig. 3 The balance performance of 16/900

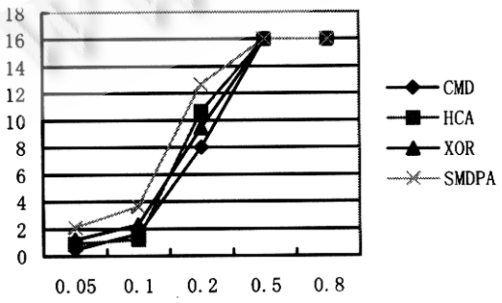

Fig. 4 The parallel performance of 16/2500

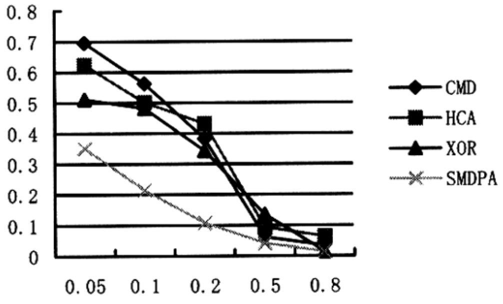

Fig. 5 The balance performance of 16/2600

© 中国科学院软件研究所 http://www.jos.org.cn

From these figures we can find that the SMDPA algorithm has better parallel and balance performance than other three algorithms in all simulation conditions. For example, when there are 32 PUs and 2500 data hyper-cubes in the system, SMDPA's DP value is 27.216 when the length of range query is 0.5, and the DB value is 0.0851. At the same time the Hilbert Curve algorithm's performance values are 15.216 and 0.2154.

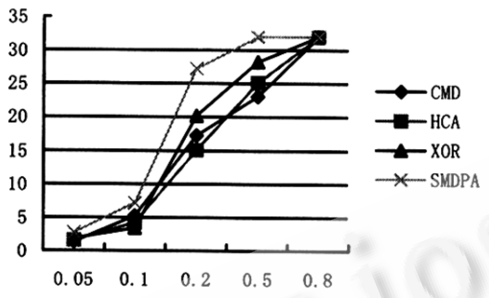

Fig. 6 The parallel performance of 32/2500

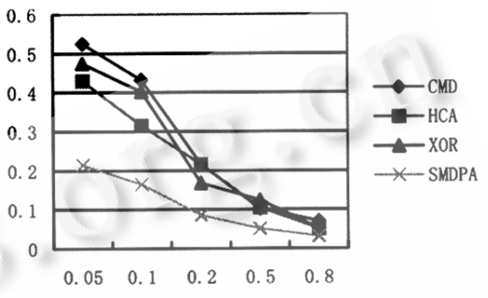

Fig. 7 The balance performance of 32/2500

And when the number of PUs increases and other condition is constant, we can find that the SDMPA algorithm is better. For example, when the number of data hyper-cubes is 2500 and the length of range query is 0.2, if the number of PUs is 16, SMDPA can achieve the parallel performance of 12.688 which just is 0.793 of 16, and the balance performance value is 0.1065. But when the number of PUs is 32, the parallel performance is 27.216 and 0.8505, and balance performance is 0.0851. It can be explained that when the PUs' number increases the SMDPA can place the data hyper-cubes in the same linked list among different PUs. That is good for data range query since the range query has locality character.

When the number of data hyper-cubes increases and other condition is constant, we can find that the SMDPA is better. For example, when the number of PUs is 16 and the length of range query is 0.2, if the number of data hyper-cubes is 900 the SMDPA algorithm's parallel performance is 11.357 and balance performance is 0.1354. While number of data hyper-cubes is 2500, the performance is 12.688 and 0.1065. The reason is as follows. When the number of data hyper-cubes increases, the information about the hyper-cubes' PAF and similarity is more sufficient, then the SMDPA can place the data hyper-cubes more accurately and effectively. As a result the data range query will have better performance.

4 Conclusion

In this paper we describe the SMDPA algorithm, which can be very effective to place the multidimensional data hyper-cubes even they have different accessed frequencies. It uses the information about the hyper-cubes and similarity between them to organize all hyper-cubes to some sub-graphs. The hyper-cubes in one sub-graph will be accessed together with high frequency. We can use two methods to place the hyper-cubes in one sub-graph and hyper-cubes that are in different sub-graphs. As a result the parallel and balance performances of the SMDPA algorithm are better than traditional algorithms. And we simulate the SMDPA to compare it with CMD/HCA/XOR algorithms. The simulation results prove that SMDPA is a good algorithm too.

Now we just place the hyper-cubes that are in different sub-graphs with different start PU IDs. We will try to find another more effective method to do that in future. And we will formalize the SMDPA algorithm and try to analyze its performance using mathematic tools.

© 中国科学院软件研究所 http://www.jos.org.cn

References:

[1] Du, J.S. Disk Allocation for Cartesian Product File of Multiple Disk System. ACM TODS, 1982. 82～101.

[2] Li, J.S. CMD: a multidimensional declustering method for parallel catabase system. In: Proceedings of the 18th International Conference of Very Large Data Bases. Vancouver, Canada, 1992. 23～27.

[3] Kim, S.P. Optimal file distribution for partial match queries. In: Proceedings of the ACM SIGMOD. Chicago, Illinois, 1988. 173～182.

[4] Faloutsos, D.M. Disk allocation using error correcting code. IEEE Transactions on Computers, 1991,40:907～914.

[5] Faloutsos, C. Declustering using fractals. In: Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems. San Diego, CA, 1993: 20～23.

[6] Srivastava J., *et al*. Data declustering in PADMA: a parallel database manager. Data Engineer, IEEE Computer Society, 1994,17(3):3～13.

[7] Bell D. A. Difficult data placement problem. The Computer Journal, 1984,27(4):315～320.

一种新型多维数据空间放置算法

谭郁松[1]， 杨 利[2]， 周兴铭[1]

[1](国防科学技术大学 并行与分布处理国家实验室,湖南 长沙 410073);

[2](东北大学阿尔派公司 计算机软件中心,辽宁 沈阳 110006)

摘要: 介绍了一种新型的多维数据空间放置算法——SMDPA.该算法使用数据超方体的先验被访问概率以及访问之间的相似度放置超方体.即使在数据超方体的被访问频率不满足均匀分布的情况下,该算法可以有效地放置超方体.模拟结果证明,该算法较传统算法有更良好的性能.

关键词: 多维数据空间;相似度;并行数据库

中图法分类号: TP311 **文献标识码**: A

www.jos.org.cn