并行面向对象数据库中的数据放置*

王意洁　胡守仁

(国防科技大学计算机系　长沙　410073)

摘　要　针对并行面向对象数据库的特点, 结合异步并行查询执行策略, 提出了一种基于对象类的混合式数据放置策略, 它包括两个部分: 混合式数据划分策略和基于对象类的数据分配策略。

关键词　并行面向对象数据库, 异步, 数据放置, 数据划分, 数据分配

分类号　TP311.132.4

Data Placement in Parallel Object-Oriented Database

Wang Yijie　Hu Shouren

(Department of Computer, NUDT, Changsha, 410073)

Abstract　In this paper, according to the features of parallel object-oriented database and asynchronous parallel query execution strategy, a class-based hybrid data placement strategy is proposed, which consists of two parts: the hybrid data partition strategy and the class-based data allocation strategy.

Key words　parallel object-oriented database, asynchronous, data placement, data partition, data allocation

In the shared-nothing multiprocessor environment, data parallelism is an important parallelism of POODB, which requests that the database is partitioned and allocated on several processors, this is always named data placement. Data placement[1,2] consists of data partition and data allocation. Data partition is to partition the database into several data subsets, data allocation is to allocate these data subsets onto processors. Data placement is a critical factor in achieving good system performance, and it is associated with the parallel processing strategy of POODB.

1　Class-Based Hybrid Data Placement Strategy

Though asynchronous parallel query execution strategy is able to avoid the unnecessary data propagation efficiently, the communication overhead is still a critical factor which affects the system performance. According to the essential features of asynchronous parallel query execution strategy, the principles of data placement are proposed:

(1) improve the speed of accessing data;

(2) exploit all kinds of parallelisms efficiently;

(3) localize retrieval and manipulation of data;

(4) reduce the overall communication across the processing nodes;

(5) utilize the CPU resource reasonably.

Based on the above consideration, the class-based hybrid data placement strategy is proposed, which consists of two parts: the hybrid data partition strategy and the class-based data allocation strategy.

2　Hybrid Data Partition Strategy

There are two kinds of traditional data partition strategies: horizontal partition strategy and ver-

* 国家部委项目资助, 国家自然科学基金资助项目
1999年2月26日收稿
第一作者: 王意洁, 女, 1971年生, 博士生

tical partition strategy. The horizontal partition strategy is suitable for exploiting inter-object parallelism, while the vertical partition strategy is suitable for exploiting intra-object parallelism. The horizontal partition strategy is essential for parallel relational databases to achieve good salability and speedup, it has the advantage that a selection operation can be carried out very efficiently since its processing time is the time for processing the largest segment instead of the entire relation, but it may not be suitable for processing of multiple concurrent object-oriented queries. The vertical partition strategy is suitable for processing of complex objects, but it will not scale up well.

According to the principles of data placement, the hybrid data partition strategy is proposed: Firstly, the objects are vertical partitioned, there are two types of partitions: (a) data values stored in Oid-data value pairs, (b) object cross-references stored in Oid-Oid pairs. Secondly, the vertical partitions are horizontal partitioned based on Oid, the resulted partitions are called hybrid partitions. The hybrid partition consists of Oid-data pairs and Oid-Oid pairs, and all data of an object is stored in a hybrid partition. The hybrid partition strategy can avoid many unnecessary I/O requests in accessing complex objects, improve the speed of accessing data, and provide opportunities for exploiting all kinds of parallelisms.

3 Class-Based Data Allocation Strategy

3.1 The Goal and Principles of Data Allocation

The goal of data allocation is : the loads of processors are balance, the communications among processors are lowest. The principles of data allocation are: (1) Balancing the working load of processors; (2) Reducing the communication overhead among processors.

3.2 The Overview of Class-Based Data Allocation Strategy

According to the goal and principles of data allocation, the class-based data allocation strategy is proposed: Firstly, based on the load of every class, the communications among classes, the number of classes and the number of processors, the classes are combined or partitioned reasonably, so several class sets are constructed, the loads of class sets are balance, the communication overhead among class sets is lowest, and the number of class sets is equal to the number of processors; Secondly, the class sets are allocated onto processors, the communication overhead among processors is lowest.

3.3 Description of Constructing Class Sets Algorithm

The first step of class-based data allocation strategy is to construct a number of class sets.

Constructing Class Sets Algorithm:

(The number of processors is N ; the number of classes is M; C_i is an object class ($i = 1, 2, \cdots, M$))

If $M = N$, *then every class constructs a class set*;

If $M > N$, *then combine the classes*:

(1) According to the loads of classes, the classes are ordered decreasely;

(2) Compute the average load of class sets: $SetWorkAve = \sum_{i=1}^{M} Work(C_i)/N$

(3) Initialize M class sets (each one is empty);

(4) Every class whose load is not less than $SetWorkAve$ constructs a class set. If there are m such classes, then the rest classes will be placed into the rest (M-m) empty class sets.

(5) Compute the average load of class sets again: $NewSetWorkAve = \sum_{i=m+1}^{M} Work(C_i)/(N - m)$

(6) For $i = m + 1$ to M Do

(a) Try to place C_i into every class set, if the sum of C_i's load and a class set's load is not more than $NewSetWorkAve$, then this means that C_i can be placed into this class set;

(b) If the class sets which C_i can be placed in are all empty, then place C_i into the first class set;

(c) If the class sets which C_i can be placed in are not all empty, then according to the communication between the unempty class sets and C_i order the unempty class sets decreasely, place C_i into the first unempty class set. If there are more than one class sets whose communications are biggest, then select a class set whose load is bigger and place C_i into it;

(7) If there are L unempty class sets in $(M-m)$ class sets and $L>(N-m)$,
then (a) Order the unempty class sets decreasely according to the load of them;
(b) For $j=N-m+1$ To L Do
For every class in No. j unempty class set Do
(a) Find a class set with the least load from the first $(N-m)$ unempty class sets, fetch the class and place it into this class set.
(b) If there are more than one class sets with the least load, then select a class set whose communication with the class is the biggest, and place the class into it.
(c) Recorder the first $(N-m)$ unempty class sets decreasely according to the load of them.

· *If $M<N$, then partition the classes*:

(1) According to the loads of classes, the classes are ordered decreasely;

(2) Partition the class whose load is the biggest into two hybrid partitions whose loads are equal. At this time, if the number of classes and hybrid partitions is N, then every class and hybrid partition constructs a class set; else reorder the classes and hybrid partitions decreasely according to the load of them;

(3) If there is a class whose load is the biggest among all classes and hybrid partitions, then repeat step2; else there is a hybrid partition whose load is the biggest among all classes and hybrid partitions, repartition the class which the hybrid partition belongs to: suppose that the class had been partitioned into k hybrid partitions, now repartition the class into $(k+1)$ hybid partitions whose loads are equal. At this time, if the number of classes and hybrid partitions is N, then every class and hybrid partition constructs a class set; else reorder the classes and hybrid partitions decreasely according to the load of them;

(4) repeat step 3 until the class sets are constructed.

3.4 Description of Allocating Class Sets Algorithm

The second step of class-based data allocation strategy is to allocate the class sets constructed in first step onto the processors.

Allocating Class Sets Algorithm

(1) *Preprocess*:

· According to the communication among class sets, the connections among class sets can be classified into two types: high-communication connection and low-communication connection;

· Determine the allocation priorities of class sets: the class set has more high-communication connections, its allocation priority is higher; If there are two class sets which have same number of high-communication connections, then the one with more low-communication connections has higher allocation priority; If there are two class sets which have not only same number of high-communication connections but also same number of low-communication connections, then determine their allocation priorities according to the numbers of two types of connections of their adjacent class sets;

· Build the multilevel structure of class sets:

(a) The class sets with the highest allocation priority form the first level;

(b) The adjacent class sets of class sets in Level k form level $k+1$; if one of them has been in first k levels, then it cannot be placed in level $k+1$;

(c) Order all class sets in every level decreasely according to their allocation priorities;

· The class sets without high-communication connections are marked by a delay-token;

· Determine the priorities of processors: A processor with a large number of connected processors is assigned a higher priority; The priorities of two processors with the same number of connected processors are determined based on the number of the connected processors of all their connected processors;

(2) *According to the multilevel structure of class sets, allocate the class sets from level* 1 *to the last level*:

· Allocate the class sets in level 1 onto several processors with the highest priority;

· For every class set without delay-token in level $k+1$, firstly, find the processors of its adjacent class sets in level k; secondly, find these processors' connected empty processors; lastly, allocate the class sets with higher allocation priority onto the connected empty processors with higher priority.

· For every unallocated class set without delay-token in level $k+1$, allocate it according to the association between it and the allocated class sets;

(a) If there are some high-communication connections between the unallocated class set and the allocated class sets, and the processors of these allocated class sets have some connected empty processors, then select an allocated class set with the highest priority, find a processor with higher priority from its connected processors, and allocate this unallocated class set onto it;

(b) If there are some low-communication connections between the unallocated class set and the allocated class sets, and the processors of these allocated class sets have some connectd empty processors, then select an allocated class set with the highest priority, find a processor with higher priority from its connected processors, and allocate this unallocated class set onto it;

(c) Else, allocate the unallocated class set onto the close processor of its adjacent class sets in level k; (3) *Order the class sets with delay-token decreasely according to the priorities of them*:

If there are some low-communication connections between the class set with delay-token and the allocated class sets, and the processors of these allocated class sets have some connectd empty processors, then select an allocated class set with the highest priority, find a processor with higher priority from its connected processors, and allocate this class set with delay-token onto it;

(4) *Allocate the unallocated class sets with delay-token onto the rest empty processors, the class set with higher priority is allocated onto the processor with higher priority.*

4 Conclusion

Data placement is an important research topic in the field of parallel database. After the research of traditional data placement strategies, according to the features of POODB and asynchronous parallel query execution strategy, the reasonable principles of data placement are proposed. Based on the principles, the class-based hybrid data placement strategy is proposed and implemented. Experimental results show that the class-based hybrid data placement strategy can overcome the limitations of traditional data placement strategies, it is a practical strategy.

References

1 Bassiliades N, Vlahavas I. A non-uniform data fragmentation strategy for parallel main-memory database systems. In Proc. 21th Int'l Conf. on Very Large Databases, Switzerland, 1995: 370～381

2 Haddleton R F, Pfaltz J L. Parallelism in Scientific Database Queries. Proceedings of the Eighth International Working Conference on Scientific and Statistical Database Management, Stockholn, 1996: 101～126