Logical Data Partitioning for Shared-disk Database Cluster

SUI Xin-zheng
College of Information Science
Nankai University
Tianjin, China
xinzhengs@gmail.com

CHENG Ren-hong
College of Software
Nankai University
Tianjin, China
chengrh@nankai.edu.cn

***Abstract*—In a shared-disk database cluster, the overhead of data synchronization between nodes cannot be ignored, especially when the speed of the interconnect network is not fast enough. It is more likely to become the system bottleneck. However, most of those overhead can be avoided. This paper presents a logical data partitioning method. It can decoupling the nodes and reduce the overhead of interconnect network communication by using a data-oriented task dispatching strategy. Because the partitioning method is logical and it is easy to change for avoiding the data skew. The experiment results show that the proposed method can greatly reduce the communication data amount and improve the system performance.**

Keywords-logical data partitioning; load balancing; database; cluster; shared-disk; shared-storage

I. INTRODUCTION

In shared-disk database cluster, each node in it has private CPU and memory, and share a storage system which keeps the data of the whole database. This architecture is shown in Figure 1(a). When system is processing tasks, each node will load the data needed from disk to memory through a network. Because the I/O speed of disk is much slower than the main memory. In order to prevent the extra disk I/O, the modified data is not written back to the disk immediately, in case that the data will be used again in future. However, in shared-disk database cluster, different nodes may cache the same data. In order to ensure data consistency, the node that has the latest data should write it back to the shared storage system to make sure other nodes can get the latest data. So the shared storage system became the communication media of all the nodes in the cluster. When the concurrent requests on the same data are very frequent, there will be a lot of disk I/Os to keep the data consistency. This will reduce the system performance.

In order to solve this problem, Oracle proposed the Cache Fusion technology [1]. The node that has the latest copy of data does not have to write it to the shared storage system for other nodes. But directly send the data through an interconnect network. This can avoid a lot of disk I/O operations. However, the network traffic will be increased. This technology also needs a high speed network to keep the system performance in a reasonable level. In the general business environment, there is still a wide gap between network and memory speed. The network bandwidth may become the new bottleneck of system performance.

978-1-4244-8728-8/11/$26.00 ©2011 IEEE

In shared-nothing database cluster, the nodes are connected by a network and each node has its own CPU, memory and disk, as shown in Figure 1(b). The whole data of the database is distributed across the disks of each node. In such system, the physical partitioning method is very important to decouple data and improve the system parallel processing ability. There are many data partitioning methods, like round-robin, hash, range [2], Hybrid-Range [4] [5]. But when the physical partition is made, it is relative hard to change.

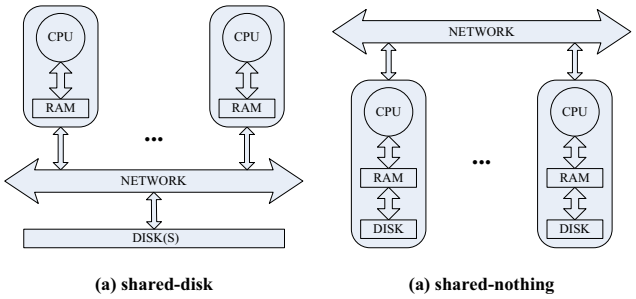

Figure 1. Two classic database cluster architectures.

In shared-disk or shared-nothing system, data is processed in the node's private memory. The only difference between them is the place where the data is stored. It is possible to make a logical partitioning on a shared-disk system and make it looks like a shared-nothing system. So it will have benefit from both architectures.

The remainder of this paper is organized as follows: Section 2 discusses the logical partitioning method in detail and section 3 we build a connection pool framework which can distribute the tasks by a data-oriented strategy. Section 4 compares the proposed method with the random task distributing strategy and the TPC-C benchmark is used to show the effect. Section 5 provides the conclusion and future work.

II. THE LOGICAL DATA PARTITIONING METHOD

Logical data partitioning (LDP) should be in accordance with the characteristics of the tasks. The tasks that access the same data should be allocated to the same node, which can avoid the overhead of data communication between nodes. Take a course-selecting system as an example: students with the same majors are more likely to read the same data. So it should be send to the same node.

To find the proper partitioning attribute (PA) is very important. It can not only avoid the data skew but also decouple the nodes. The PA can be determined by the knowledge of the business logic. And it also can be determined by the relationship between the tables. We will discuss the second one in detail.

For a well designed database, the foreign keys between tables indicate some kind of relationship exists. If a table T_S has a foreign key K_S, it referenced the key K_M of table T_M. We call the K_M is a master key of K_S and Ks is a slave key of K_M. The K_M may also be a foreign key. So K_M has its own master key. If K_M is not a foreign key, K_M is a root key of K_S. If there is no key from other tables referenced the K_S, it is called a leaf key. There may be more than one leaf key across the tables and each leaf key can form a tree structure based on the key relationship. If the root key is partitioned, the tables of slave keys are also being partitioned. So the PA should come from the root node of the tree with the max number of nodes. The foreign key-based PA searching algorithm is showed in Figure 2. The record is used to store the root keys. The one with the maximum weight is the PA. Then we can be used a range partitioning method to partition the tables of the tree.

```
1.  PA_SEARCH() {
2.    for each table t in T {
3.      for each foreign key k in t {
4.        while have_master_key(t, k) {
5.          tmp = get_master_key(t, k);
6.          t = tmp.table;
7.          k = tmp.key;
8.        }
9.        if record(t, k) exists {
10.         record(t, k).weight++;
11.       }
12.       else {
13.         record.add(t, k);
14.         record(t, k).weight=1;
15.       }
16.     }
17.   }
18. }
```

Figure 2. The PA searching algorithm.

We take the TPC-C benchmark as example to demonstrate the PA searching method. There are nine tables and five transactions in TPC-C. The NEW_ORDER and PAYMENT are the two main transactions which take up 55% and 43% each. These two transactions visit all the nine tables. So T = {CUSTOMER, DISTRICT, HISTORY, ITEM, NEW_ORDER, ORDER_LINE, ORDERS, STOCK, WAREHOUSE}. The keys referenced relationship is described in TPC-C benchmark document [3]. Because all the tables are have primary or reference keys. In TABLE I, the search result is listed. The highest weight is WAREHOUSE.W_ID, and it should be chosen as the PA.

TABLE I. THE PA ALGORITHM SEARCH RESULT

Root Keys	**Search Result**	
	Slave Keys	***Weight***
WAREHOUSE.W_ID	STOCK.S_W_ID ORDERS.O_W_ID ORDER_LINE.OL_W_ID NEW_ORDER.NO_W_ID HISTORY.H_W_ID DISTRICT.D_W_ID CUSTOMER.C_W_ID	7
DISTRICT.D_ID	ORDERS.O_D_ID ORDER_LINE.OL_D_ID NEW_ORDER.NO_D_ID HISTORY.H_D_ID CUSTOMER.C_D_ID	5
CUSTOMER.C_ID	ORDERS.O_C_ID HISTORY.H_C_ID	2
ORDERS.O_ID	ORDER_LINE.OL_O_ID NEW_ORDER.NO_O_ID	2
ITEM.I_ID	STOCK.S_I_ID ORDER_LINE.OL_I_ID	2

III. THE CONNECTION POOL FRAMEWORK

In order to distribute the tasks across the nodes we use a connection pool framework. The framework uses a modular design pattern and resides in the client side. The framework structure is shown in Figure 3.

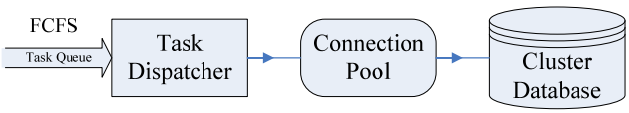

Figure 3. The connection pool framework.

1) Task Dispatcher

Based on the strategy provided, the task dispatcher distributes tasks of the queue to the nodes in the cluster. This is done by providing the tasks with connections that linked to different nodes.

2) Task Queue

The task queue plays a role in buffering the tasks waiting for processing. The queue is running in first come first serve (FCFS) order.

3) Connection Pool

The connection pool maintains many connections which point to different nodes in the cluster. It provides two basic operations which are getting and releasing connections.

4) Database Cluster

The database cluster is composed of multiple nodes. It appears as if one server to the outside world. The nodes are connected by a network.

IV. EXPERIMENTS AND EVALUATION

The workload is generated from the TPC-C benchmark and we make a little change. We use stored procedures to implement the five transaction classes. The isolation level of transactions is "Serializable". Each transaction is a task. We assume the arrival process of tasks is a Poisson process. Let λ as the intensity of the homogeneous Poisson process $\{N(t) \mid t \geqslant 0\}$, where $\lambda > 0$. The task arrival interval sequence is $\{T_n,$

n> 0}. At this point, T_1, T_2, ... , T_n are independent and obey exponential distribution of parameters λ. Random(0,1) means the uniform random number of interval (0,1). So the task arrival time interval can be expressed as:

$$T_n = -\frac{\ln(Random(0,1))}{\lambda}$$

At the beginning of the test, there is no data cached in the memory of the nodes. So physical read operations will be performed. This means the task processing will be very slow. And the next coming task is probably blocked in the waiting queue. As the data ready in the memory, the task processing is getting fast. But there many tasks are waiting in the queue. It looks like all the tasks coming without an interval and the task generation simulation is failed to take any effect. So before the test begins, a warm up is necessary.

The experiment consists of an application client, a database server (two nodes) and a disk server (provides shared disks for database server). These servers are connected by private network through a 1000M Ethernet switch. The public network is build by a 100M Ethernet switch by which those servers can be connected by the application client. The computers specific configuration is shown as follows:

1) Application Client: Windows XP SP3 (32bit), Lenovo M5100t, AMD PhenomII X4 810 3.0GHz, 4GB RAM, 320GB HDD.

2) Database Server: Solaris 10 (64bit), Oracle 11g RAC, Lenovo M5100t, AMD PhenomII X4 810 3.0GHz, 4GB RAM, 320GB HDD.

3) Disk Server: Ubuntu 10.04 Server (32bit), HP ProLiant ML350 G3, Intel Xeon 2.8GHz, 1GB RAM, 72G+36G*3 SCSI.

We initialize the connection pool with 50 connections for each node (node1 and node2) and use the parameter λ of 50 to generate 5000 tasks. The five transactions are NEW ORDER. PAYMENT, ORDER STATUS, DELIVERY and STOCK LEVEL, the mix rate is 55:43:4:4:4. The data is generated followed the TPC-C specification. There are 10 warehouses. The probability of transactions generated under each warehouse is equal. The experiments are described as follow:

- Exp1: The task dispatcher uses random strategy to distribute tasks to node1 and node2. Each node has the equal possibility to get a new task.
- Exp2: The task dispatcher distributes tasks according to the different warehouse id. We use a simple hash function to do this: node_id = warehouse_id % 2 + 1, node_id is the node that the task will be assigned, warehouse_id is the identity of the warehouse; '%' is modulo operation. The distribution of warehouse_id is uniform. So each node has the equal possibility to get a new task.

We conducted two times for each experiment, one for warm-up and another for result. TABLE II shows the mean execution time of five transactions of the two dispatching strategies. We can see that for each transaction, the LDP strategy is much quicker than the random strategy. The average improvement of execution time is about 81%.

TABLE II. THE AVERAGE EXECUTION TIME OF DIFFERENT DISPATCHING STRATEGIES

Transactions (Count)	Dispatching Strategy	
	Random	***LDP***
NEW_ORDER (2250)	0.39s[a]	0.14s
PAYMENT (2150)	2.06s	0.28s
ORDER_STATUS (200)	0.12s	0.07s
DELIVERY (200)	0.30s	0.13s
STOCK_LEVEL (200)	0.67s	0.13s
SUMUP (5000)	1.10s	0.20s

a. 's' means seconds.

TABLE III shows the disk and network data amount usage of the two dispatching strategies. All the resource usage of LDP strategy is much less than the Random strategy. For LDP, most data is processing in the memory and there are less disk I/Os or network communications. That's why the tasks are executed faster than the random strategy.

TABLE III. THE DATA AMOUNT USAGE OF DIFFERENT DISPATCHING STRATEGIES

System Resource Usage	Dispatching Strategy	
	Random (node1 / node2)	***LDP (node1 / node2)***
Physical Read	68.5MB[a] / 79.1MB	48.1MB / 21.1MB
Physical Write	5.2MB / 21.1MB	0.6MB / 0.04MB
Network Send	231.0MB / 185.6MB	92.4MB / 37.8MB
Network Receive	58.9MB / 48.7MB	21.0MB / 19.7MB

a. 'MB' means mega bytes.

V. CONCLUSION AND FUTURE WORK

In order to reduce the communication overhead between nodes and improve the parallel processing ability of the shared-disk database cluster, we propose a logical data partitioning method and build a connection pool framework which can distribute the tasks according the data. The experiment results show that the proposed method can not only reduce the communication overhead across nodes, but also can increase the task processing speed. But we only use the uniform distribution of partitioning attribute as the experimental tasks. For the non-uniform distribution, there is no dynamic regulation ability. How to make dynamic adjustment of logical data partitioning according to the tasks is the next step work.

ACKNOWLEDGMENT

This work is supported by the China Academic Degrees & Graduate Education Development Center under contract number CDGDCPGC2009K1.

REFERENCES

[1] T. Lahiri, V. Srihari, et. al., “Cach Fusion: Extending shared disk clusters with shared caches,” Proc. 27th VLDB conference, Rome, Italy 2001.

[2] David DeWitt, Jim Gray. “Parallel database systems: the future of high performance database systems,” Communications of the ACM, 1992, 35(6): 85~98.

[3] Transaction Processing Performance Council, “TPC Benchmark C, Standard Specification, Revision 5.11,” February 2010, http://www.tpc.org/tpcc/spec/tpcc_current.pdf

[4] S.Ghandeharizadeh, J.D.Dewitt, “Hybrid-range partitioning strategy:a new declustering strategy for multiprocessor database machines,” In Proceedings of the 16th VLDB Conference. Palo Alto: Morgan Kaufmann, 1990.481~492.

[5] K.Q.Nguyen, T.Thompson, G.Bryan, “An enhanced hybrid range partitioning strategy for parallel database systems,” In Proceedings of the Eighth International Workshop on Database and Expert System Applications.Los Alamitos: IEEE Computer Society, 1997.289~294.