TELKOMNIKA, Vol. 11, No. 4, April 2013, pp. 1896~1901
ISSN: 2302-4046

An Improved-hash Based Multi Dimensional Distributed Index Mechanism

Haiwen Han*[1, 2], Deyu Qi[1], Weiping Zheng[3]
[1]Research Institute of Computer Systems, South China University of Technology, Guangzhou, China
[2]College of economic and management, South China Normal University, Guangzhou, 510631, China
[3]College of computer science and Engineer, South China Normal University,510631 Guangzhou, China
*Corresponding author, e-mail: hanhw@scnu.edu.cn

Abstract

Data partition and the accordingly index technologies which could result in uniform data distribution and fast data finding are critical in high parallelism for shared nothing architecture to minimize the transaction processing time. An improved-hash based multi dimensional index mechanism is present in this paper to achieve high parallelism performance for distributed data-parallel computation in shared nothing architecture. After partitioning and storing data using improved-hash function based on partitioning column, the multi dimensional indexes based on multi columns and the corresponding data lookup procedure are constructed. Afterwards, the space complexity and time complexity are analyzed.

Keywords*: data partition, hash index, shared-nothing architecture, distributed index*

1. Introduction

In shared nothing architecture shown as Figure 1, the primary processing node is responsible for receiving and passing the query transaction to processing node(s) to finish the transaction in parallel using their own independent components including processing unit, memory unit and storage unit [1]. So, this shared-nothing architecture effectively raises up parallelism and scalability performance for distributed data-parallel computation. Being used to divide large table into smaller and manageable pieces, data partitioning and the accordingly index technologies play important role in the implementation of distributed storage and parallel processing in this architecture. The traditional range data partitioning approach dividing table rows to partitions based on value range of partitioning column is easy to perform. But the accordingly index does not adapt to the real-time application because of the slow data lookup speed and the non-uniformly data distribution [2]. The hash data partitioning approach could fasten the data lookup speed by mapping partitioning column value to short hashed key and constructing hash index to achieve quit hash search. But still this approach and index could not ensure average data distribution [3]. And, both the range data partitioning approach and the hash data partitioning approach focus only on one table column doesn't provide a multi dimensional index mechanism.

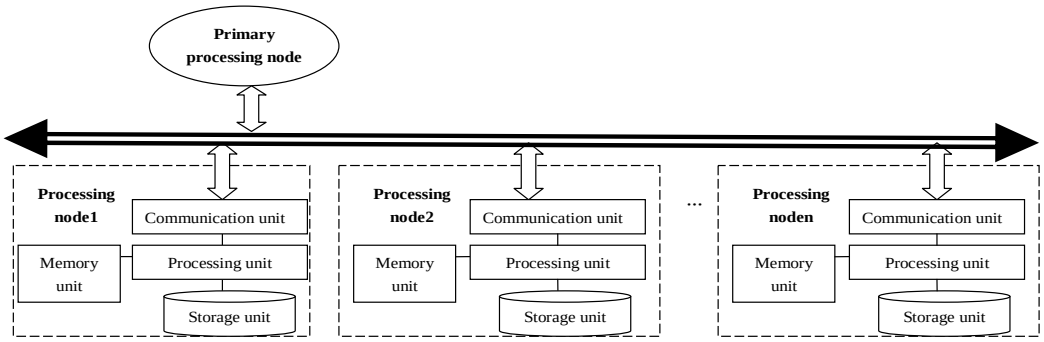

Figure 1. Shared-nothing Architecture

Received January 10, 2013; Revised February 10, 2013; Accepted February 26, 2013

This paper brings up an improved-hash based multi dimensional distributed index mechanism to achieve the uniformly data distribution and fast data lookup speed for data parallel computation. This mechanism implements a uniformly data storage among multiple processing nodes by using hash function to divide massive data into hash buckets and using bucket adjustment algorithm to assign buckets to different processing nodes. Accordingly, distributed index composed of one primary hash index in primary processing node based on partitioning column and a set of ordered index in each processing node based on other column(s) is constructed. High performance in parallelism and transaction processing speed would be achieved by this partitioning column based uniformly data distribution and the accordingly multi dimensional distributed index mechanism.

2. The Index Mechanism

Assuming that there are N processing nodes, relation R, partitioning column R.x with value range V.

2.1. Data Partition

Being decided by user need, the partitioning column is composed of one or more attributes in relation R. A hash function $H:V\to\{0,1,\cdots,M-1\},(M\geq N)$ is given to partition relations R into M buckets based on the partitioning column R.x. T(R,R.x,I) is a data bucket numbered I. For any tuple r in R, T(R,R.x,I)= H(r[R.x]).

A bucket assign function $F:\{0,1,\cdots,M-1\}\to\{0,1,\cdots N-1\}$ is given to assign the M buckets to N processing nodes' storage units. Accordingly, for any tuple r in R, F(H(r[R.x])) is the processing node number of r.

2.2. Data Storage and the Accordingly Primary Hash Index

Assuming that there are N processing nodes, relation R, partitioning column R.x with value range V.

It is important to get uniform data distribution by hash function. But even a good hash function (like mention [4]) couldn't achieve totally uniform data distribution among buckets and processing nodes. So it is needed to use bucket average algorithm to adjust the bucket's storage location.

Without loss of generality, assuming that M=256, N=32, F(x)=x mod N, then as Figure 2 shown, the p0 node would be assigned 8 buckets numbered 0,32,64,96,128,164,196,224 and sorted by their tuples number. And other processing nodes also have buckets of tuples. Bucket average algorithm would firstly visit every node to sort buckets into a queue $T_{i0},T_{i1},\cdots T_{i255}$ by their tuples number. And then, data exchanges between bucket pairs T_{i0} and T_{i255}, T_{i1} and T_{i254}…, and so on are implemented to uniform the tuples number among the processing nodes. At the same time, as Figure 3 shown, a primary hash index for 256 buckets and 32 nodes is accordingly constructed in primary processing node.

Bucket number Tuples

32	3
0	17
64	29
.	.
.	.
.	.
128	65

Figure 2. The bucket queue

Bucket number Node

0	16
1	29
2	31
3	2
.	.
.	.
.	.
254	23
255	18

Figure 3. Primary hash index

Rather than a absolutely uniform data distribution, a nearly uniform distribution do better in reducing the algorithm's complexity and minimizing the execution time for application. So, a threshold ε is given to measure buckets difference and accordingly the bucket pair exchange data only if their difference is more thanε(buckets' tuples number is admitted as equal if their difference is less than ε). And the primary index would also change accordingly.

The more uniform data distribution among buckets the hash function makes, the less data exchange occurs between buckets pair. A.Ostlin [5] provide a hash function which is good at making uniform data distribution among buckets.

2.3. The Distributed Ordered Indexes

The data partition and the data storage with the accordingly primary hash index discussed before are all based on the partitioning column, without loss of generality, named A. The indexes based on other non-partitioning column(s) are needed for the transactions about those non-partitioning column(s). Assuming that the non-partitioning columns named $B_1, B_2, \cdots, B_k$, and there are tuples $S_i = \{r_{i1}, r_{i2}, \cdots r_{ip}\}$ in processing node numbered i, k ordered indexes based on $B_1, B_2, \cdots, B_k$ separately are constructed in this node.

Without loss of generality, by the ordered index based on B1 shown as Figure 3, the tuples $r_{11}, r_{12}, \cdots r_{ip}$ are sorted on column B1's value $K_1, K_2, \cdots K_p$. Then the ordered index is composed of two attributes including column B1 and the corresponding pointer of tuple r's location in R.

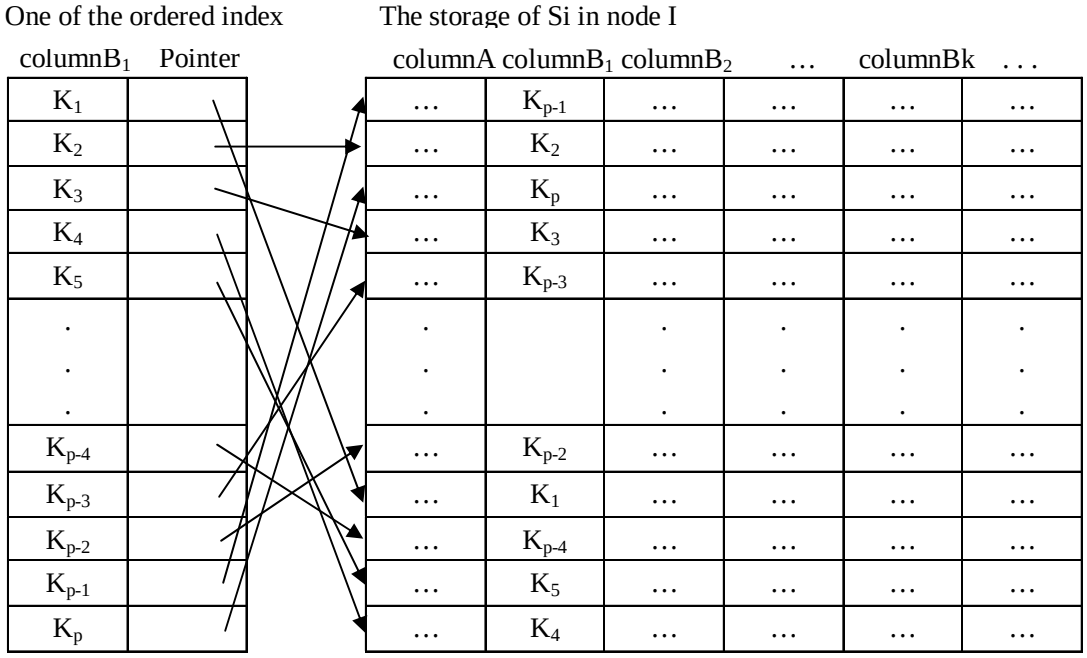

Figure 4. The Ordered Index based on B1 Column in Node I

3. The Execution Framework

In this section, the execution framework under this distributed index mechanism including the design of partition scheme and the accordingly method for data query are presented.

3.1. The Design of Partition Scheme

The table, index and index-organized are further broken down into paragraphs, through selecting the appropriate scheme. An appropriate partition scheme [6, 7] designed specially for application would make good use of the distributed index mechanism to accurately manage and

access database and increase the performance, availability and manageability of the application program. A wide variety of partition scheme is provided completely transparent for users to meet the needs of their variety of requirements. Therefore, partition can be applied to almost any application.

Considering the application of the clear time frame database [8], for example, month, quarter and year, the time field should be used as the scope partition and local partition index of partition column. The partition index is used to complete the data fast finding and improve the system responsiveness, for the majority partition column finding.

For the database applications of undefined time marked and no-listed field, it can be achieved by the hash partition [9]. Generally, the field with sequence number attribute is defined as partition column. Through the hash algorithm, the data is uniformly distributed in different I/O space of the table to improve the performance.

A trade management system [10] covers transaction processing and query operation has the large amount of data concurrent characteristics. So it is suitable for the partition application. The program objectives and principles of the partition scheme are to maintain and enhance the performance of day-to-day transactions [11]. As a typical transaction processing system, a large number of search jobs is operated by major elements, for example, timing and code. Therefore, it provides a favorable basis for partition and provide high performance by positioning the search operation to smaller units generated by partition technology.

3.2. Data Query Under the Distributed Index Mechanism

Under the index mechanism discussed, the primary processing node would analyze the query transaction and take next step according to the related columns.

1) If the query transaction relate about partitioning column A, the bucket number(s) of the corresponding tuple(s) would be counted out by hash function using column A's value in the primary processing node. Afterwards, the node(s) number(s) where those bucket(s) stored in would be counted out later by using bucket assign function and searching primary index in primary processing node. And then the query transaction would be passed to this(these) node(s) to be finished in parallel and the outcome data would be sent back to primary node at last.

2) If the query transaction relate about non-partitioning column(s) B1,B2,…,Bk, the primary processing node would firstly pass this query transaction to each processing node in parallel. By searching ordered index in each processing node using value of column Bi, processing node would get the location of tuple(s) from ordered index and get the tuple(s) from the Si storage to finish the query transaction. Outcome data would also be sent to the primary processing node at last.

It takes traditional multi dimensional index mechanism twice accesses to processing node to finish the query transaction on non-partitioning column(s). The first access to processing node is to get the node number(s) by searching index(s) in processing node(s). The second access is to pass the transaction to those node(s) found out in the first access. Being superior to the traditional multi dimensional index mechanism, the multi dimensional distributed index mechanism discussed in this paper accesses the processing node only once to pass the transaction to each processing node directly.

4. The Analysis and the Experiment

It is assuming that there are C tuples in relation R being partitioned in M buckets and stored in N processing nodes. It is also assuming that k+1 columns (A,B1,…Bk) in relation r are admitted to be used in query transaction. Then the analysis for space complexity and time complexity are shown as 4.1 and 4.2.

4.1. Analysis for Space Complexity

The buckets average tuples number is $V_m = C / M$. For primary processing node, the memory size assigned to primary index is $V_{search} = M$. For processing node, the average memory size assigned to ordered index is

$V_{index} = (k+1)V_m(M/N) = (k+1)(C/M)(M/N) = (k+1)C/N$. In addition, the storage size assigned to tuples in processing node is $V_r = V_m(M/N) = C/N$.

Accordingly, the C and k values are determined by relation size and user transaction requirement, while the M and N values could be determined artificially. The M value determined by the hash function could not be too large because little data in buckets owing to much buckets would result in more data exchanges between buckets and the large V_{search} value which means large primary index in primary processing node to spend much search time for primary index. The N value determined by the network designer should be as large as possible because that the more processing nodes would result in the shorter ordered index in processing node for spending little time to search ordered index.

4.2. Analysis for Time Complexity

The time complexity is closely related to the space complexity. Assuming that the probability of occurrence for columns $A, B_1, B_2, \cdots B_k$ are $P_0, P_1, P_2, \cdots P_k (P_0 + P_1 + \cdots + P_k = 1)$. Assuming that the average searching time for primary index is T0 and the average searching time for ordered index is T1. Then the average searching time for query transaction is $T = P_0(T_0 + T_1) + P_1T_1 + P_2T_2 + \cdots + P_kT_1 = P_0T_0 + (P_1 + P_2 + \cdots P_k)T_1 = P_0T_0 + T_1$.

The value of T0 is proportional to the value of M. Being inversely proportional to the value of N, the value of T_1 also is proportional to the values of Q and C. So, without loss of generality, it is assuming that $T_0 = M$, $T_1 = QC/N$. Then, $T = P_0M + QC/N \geq 2\sqrt{QCMP_0/N}$. It means that T would reach the minimum value when $M/N = QC/P_0$.

5. Conclusion

The distributed index mechanism present in this paper brings about high performance for user transaction by being characterized by: 1) exchanging data between buckets to get a uniform data distribution storage among multiple processing nodes. 2) resulting in effective parallelism with the nearly uniform consumption of time and space in each processing node. 3) satisfying multi types of users requirements by constructing distributed indexes based on multiple columns. 4) reducing communication workload by only one time access to processing node for transactions both on partitioning column(s) and non-partitioning columns. To improve this index mechanism only satisfying the application scene of static data storage, the next step in research is to find out a solution of application scene of dynamic data storage.

Aknowledgement

Research on Oil Electric Engine Intelligent Scheduling System under mobile base station power environment monitoring platform (No. 2011B090400622)

References

[1] Y Xu, P Kostamaa, X Zhou, et al. *Handling data skew inparallel joins in shared-nothing systems.* Proc of ACM SIGMOD'08*.* Vancouver. 2008: 1043-1052.
[2] Thorsten Schutt, Florian Schintke, Alexander Reinefeld. Range queries on structured overlay networks. *Computer Communications* . 2008; 31(2): 280-291.
[3] EA Fox, Qi Fan Chen, et al. Order preserving minimal perfect hash functions and information retrieval*. Proc of 13th Int'l Conference on R&D in Information Retrieval.* San Diego. 1990: 279-311.
[4] Zhang Chun-Xiang, SUN Li-quan. Multi-dimensional-data partition based on improved-hash method. *Journal harbin Univ. Sci &Tech*. 2001; 6(1): 24-27.
[5] Ostlin, R Pagh. Uniform hashing in constant time and linear space*. Proc of 35th Annual ACM Symposium on Theory of Computation.* 2003: 622-628.

[6] Xiangyu Zhang, Jing Ai, et al. *An Efficient Multi-Dimensional Index for Cloud Data Management.* Proc of CloudDB'09. 2009: 17-24.
[7] Meng Liu, Fang Liu, Ming-hao Tian. *Data Rapid Finding Scheme Design*. Proc of 2011 International Conference on Mechatronic Science, Electrical Engineering and Computer. 2011: 2488-2490.
[8] Qiansheng Zhang, Fuchun Liu, et al. Index Selection Preference and Weighting for Uncertain Network Sentiment Emergency. *TELKOMNIKA Indonesian Journal of Electrical Engineering.* 2013; 11(1) .
[9] Balexey Lastoyetsky, Ravi Reddy. *Distributed Data Partitioning for Heterogeneous Processors Based on Partial Estimation of Theri Functional Performance Models*. Euro-Par 2009 Workshops*. Delft.* 2010: 91-101.
[10] Ward J, Peppard J. Strategic planning for Information Systems. *West Susse: John Willey & Sons Ltd*. 2007.
[11] Mardiyono Mardiyono, Reni Suryanita, et al. Intelligent Monitoring System on Prediction of Building Damage Index using Artificial Neural Network. *TELKOMNIKA Indonesian Journal of Electrical Engineering.* 2012; 10(1): 155-164 .