Available online at www.sciencedirect.com

ScienceDirect

AASRI Procedia 5 (2013) 120 – 125

AASRI Procedia

www.elsevier.com/locate/procedia

2013 AASRI Conference on Parallel and Distributed Computing and Systems

The Method of Distribute Data Storage and Location On weights of Jump Table

Bi-lin Shao[a], Gen-qing Bian[b*], Wei-qi Zhang[a]

[a]*School of Management, Xi'an University of Architecture and Technology, Xi'an ,710055, China;*
[b]*School of Information and Control Engineering, Xi'an University of Architecture and Technology, Xi'an ,710055, China*

Abstract

Focusing on the existing location technology limited the performance of the algorithm, distributed data location strategies DLPSL based on weights of jump table is presented to solve the efficiency problem. Weights are added to nodes of jump table, so that the high rate of location of the storage node priority is found to short the search path, and improve the storage and location efficiency. System analysis shows that, the node's insertion, deleting, and location on DLPSL are more efficient than single-linked list storage structure and skip graphic. Its time complexity is *0(log n)*, and space complexity is *0(n)*.

Keywords: weights of jump table, distributed storage, data location

1. Introduction

The purpose of storing information is to access and utilize. In a distributed storage system, there are many problems, for example, existence of large amount of data and how to quickly and accurately find out the dispersed data. Usually, there are three classic index mechanism categories as following [1]: a typical representative, like extendable HASH (EH) and linear HASH (LH) and Collision chain HASH (CBH) and Controlled search multiple direction HASH (CSMH),is the index mechanism based on hash function of random data organizing; another typical index mechanism is the index mechanism of orderly organization of the data based on the query tree, such as ,B-tree, B+ tree, AVL tree, T tree, T * tree, T-tail tree, etc.; the third typical representative is a hybrid index mechanism Hybrid-HT proposed by Chanho Ryu [2],and it is a mixed mechanism combined the characteristics of hash tables with query tree. Binary search tree will produce degraded tree, although AVL search tree can ensure good performance at the same time increase the difficulty to achieve the system [3]. The jump table get a better average time complexity by randomness, and you can get a good computing performance by using jump table in dynaset and dictionary.

In the literature [4], Ma Yue et al devised a interval jump table which is based on jump table data structure for fast searching in intersectant area, analyzed the interval jump table structure principle and the basic operation process, proposed intersection area search algorithm based on interval jump table, and solved the problem that efficiency and accuracy of search algorithm is low in some of the existing dynamic region. The literature [5] proposed the concept of jump diagram

* Corresponding author. Tel.: 13319273850;
*E-mail address:*bgq_00@163.com.

doi:10.1016/j.aasri.2013.10.067

which has higher efficiency in inserting, deleting and searching, and the function of the jump diagram is similar to the jump table and the binary search tree. Hammurabi Mendes et al [6] studied the consistency of jump figure, and used synchronous lock primitives for synchronous operation to improve search efficiency in the P2P distributed environment. Ying Jun et al [7] proposed a large-scale search algorithm to solve the hand-held device calculation processing data weaker problem. John Risson et al [8] proposed search model under P2P, the index mechanism of the model is based on keyword search, which can be used for information recovery and data management.

Inspired by the jump table creation and index mechanism, this article proposed distributed data positioning strategy (DLPSL) based on weights jump table, so that the high rate of location of the storage node priority is found to short the search path and improve the location efficiency.

2. Jump table structure

Jump table is a data structure of probability, invented by William Pugh in the 1990,its list bases on the parallel link list, and the operation efficiency has significant improvement related to binary search tree (For most of operation ,the average time needs O (log n))[9]. It is almost a binary tree data structure, and its implementation is simpler than balanced tree [10, 11]. It is an extension of the linked list, and the difference between jump table and the linked list is that there are pluralities of forward pointers to achieve rapid retrieval in each node (Node) of the jump table. These pointers form multi-level pointer chains on the basis of original linked list; 0-level chain contains all pointers, 1-level chain is a subset of the 0-level chain, i-level chain is a subset of the chain of the (i-1)-level.

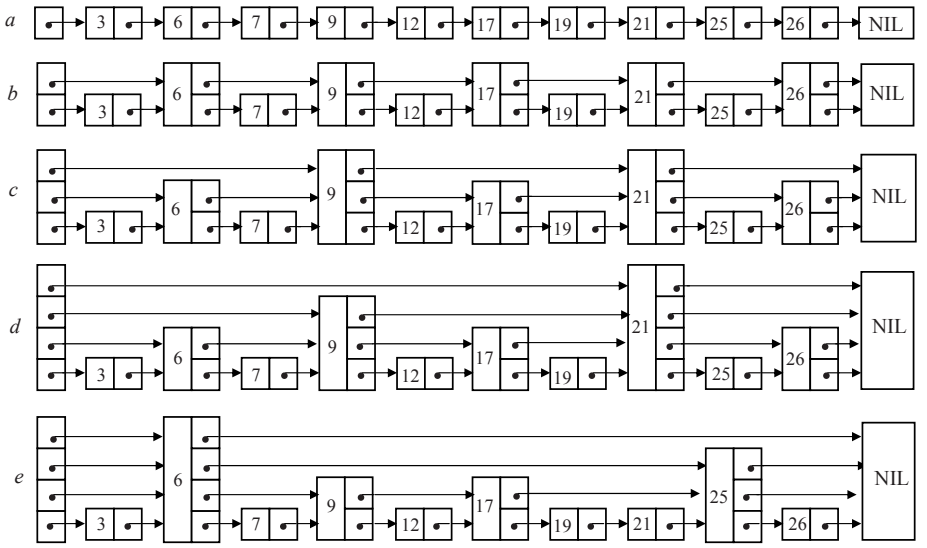

Fig.1. A different series of jump table

In the Figure 1, to find a node needs to traverse N/2 nodes in the linked list a, N/4 nodes in the linked list b, N / 8 nodes in the linked list c, and N/16 nodes in the linked list d. If there are i layers, $N/2^i$ nodes would be traversed.

The advantage of jump table is that it can skip some nodes and reduce the number of keys comparison in the retrieval process. For example, when you retrieve a jump table with two pointers, you will traverse along the first primary pointer until you find a better node value than search key, and then go back to the secondary pointer. During the retrieval process, it will start from the highest level pointer and skip as many records as needed to shorten the stride. Whether it is successful or unsuccessful, the retrieval process will skip some node value comparison; in the jump table with multi-level pointers, retrieval process will skip much more node value comparison.

3. Distributed data location strategy DLPSL

3.1 A distributed data storage system structure

As Figure 2 shown, the system consists of client, server and memory which contain one or more storage nodes.

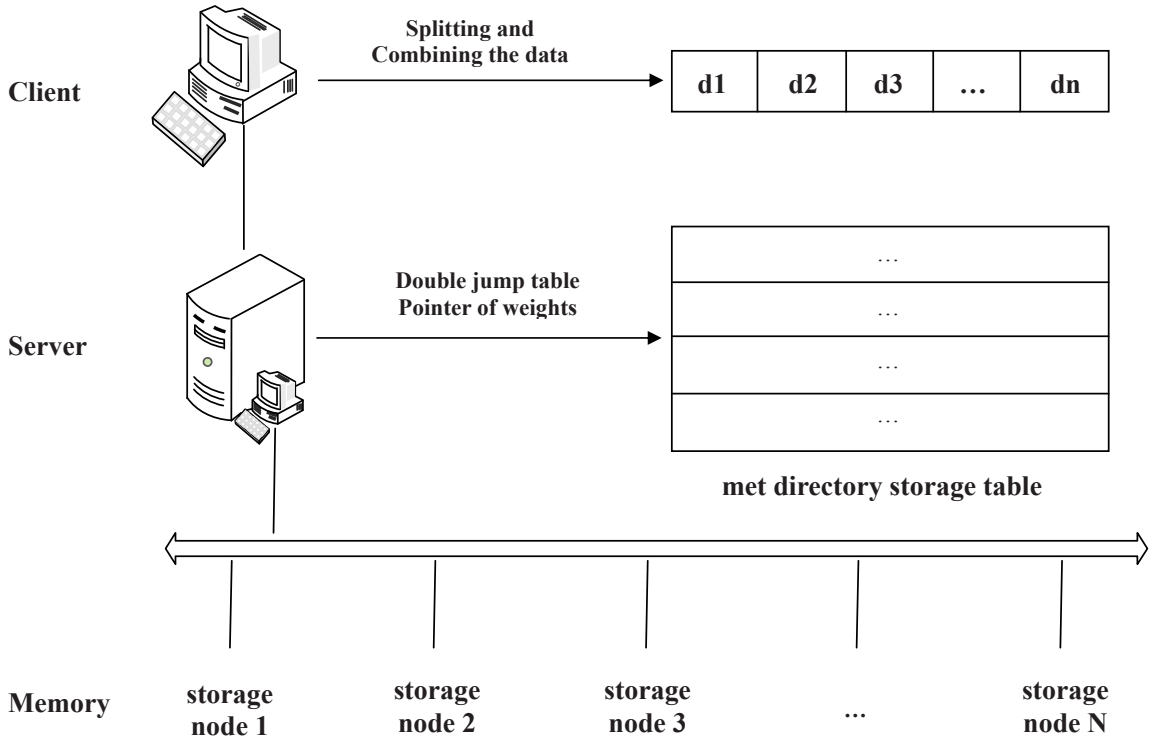

Fig.2. A distributed data storage system structure

Client: Separate the data that will be saved through a distributed storage algorithms, and then submit the separated information to the server. When the data needed to be recovered, the client sends a request to the server, calculates the raw data on itself after receiving confirmation from the server.

Server: Accept the access of the information request from the client, find data from meta directory storage table by using DLPSL quickly, and return data one by one back to the client after return the corresponding storage node.

Memory: Receive the access from the server-side I/O, each memory has a unique ID identification number (16) which is stored in the server element directory storage table.

3.2The Process of Realizing DLPSL

DLPSL adopts the double jump table storage structure which is based on weights to store nodes, and its main operations conclude: node initialization, positioning, inserting, deleting and updating of weights, etc.

• Initialization. DLPSL node is comprised of storage index value (SID), Data information (Data), storage node ID(CID), weights (priority) and the pointer field. All storage index value is often between - ∞ and + ∞, and the index value which is the basis to find the specified user data by the server which is defined as a 32 bit binary number. Data information is used to store the data of the relevant information, and set the initial value to NULL. Storage node ID is the storage ID of the corresponding file fragmentation, generally has multiple ID addresses. When you initialize the nodes, all of the weights are set to 0. The level of each node is generated by the random function, and then adjusts dynamically the weights according to the rate of searching; the layer in which the node located will dynamically change on the basis of the size of weights.

• Node Locating. When a customer access the stored data, according to the client SID which returned by the system, it uses DLPSL strategies to search on meta directory storage table. After searching SID number, corresponding CID will be return to the server end, then take out the corresponding data segment according to the CID, finally the server end will return the data fragment back to the client. Positioning node must be top-priority before deleting and inserting node, the lines in bold shows the path of query node 63 in the following figure 3.

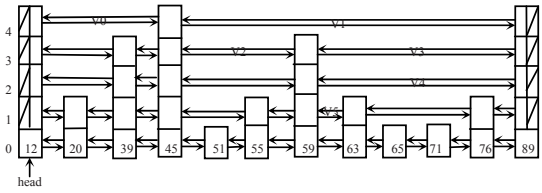

Fig.3. the searching process of node 63

```
Search(list, searchKey)
x = list->header
for i = list->level downto 1 do
while x-> pointArray [i]->key < searchKey do     //pointArray[] is Pointer Array
x = x-> pointArray [i]      and return
x = x-> pointArray [1]
if x->key = search then return x->value
else return failure
```

• Node Inserting. The main step to complete insert operation is: firstly, locate insert position of node, secondly create new nodes and finally modify the change pointer value. The insert operation will be used while updating nodes and its time efficiency mainly depends on the localization algorithm efficiency.

```
Insert(list, searchKey, newValue)
local data1[MaxLevel]      //data1[]   is the Auxiliary array
x = list->header
for i = list->level downto 1 do
while x-> pointArray [i]->key < searchKey do
x = x-> pointArray [i]
data1 [i]= x
x = x-> pointArray [1]
if x->key = search then x->value = newValue
else lal = randomLevel() //|a| is a randomly generated layer
if lal > list->level then
for i = list->level + 1 to |a| do
data1 [i]= list->header
list->level = |a|
x = makeNode(lal, searchKey, newValue)
for i = 1 to lal do
x-> pointArray [i]= data1 [i]-> pointArray [i]
data1 [i]-> pointArray [i]= x
```

• Node Deleting. Node deleting is similar to its inserting. The first is to locate the nodes that needed to delete, and then to judge whether it will find the node. If it could find the node, then modify relevant pointer value and delete node. If the node is not found, then return to failure of deleting node. The time efficiency of Node delete also depends on the localization algorithm efficiency.

```
Delete(list, searchKey)
local data1 [MaxLevel] //MaxLevel is the Max Level
x = list->header
for i = list->level down to 1 do
while x-> pointArray [i]->key < searchKey do
x = x-> pointArray [i]
--x->key < searchKey <= x-> pointArray ->key
data1 [i]= x
x = x-> pointArray [1]
if x->key = searchKey then
```

for i = 1 to list->level do
if data1 [i]-> pointArray [i]! = x then break
data1 [i]-> pointArray [i]= x-> pointArray [i]
free(x)
while list->level > 1 and
list->header-> pointArray [list->level]= NIL do
list->level = list->level - 1

• Weights Updating. When the storage node is inquired by once, its weight value will add 1.When updating by DLPSL, it judges the weights of first layer nodes whether their weights are more than or equal to 5, if node right value is greater than 5, the node series will increase 1 and the node weight be set to zero. By this way, combined with the property that series high node in jump table can be preferential to identify, it can guarantee that the node of high rate of positioning would be quickly found out, thus reduces search time.

Updata(list , searchKey)
If x->key = searchKey then
priority ++
if priority >=5
then priority=0 and priority++

4. Demonstration Analyses

4.1 Time Complexity

The probability impact factor p are introduced (p takes 1/2 usually), and p is between the pointer i and pointer i+1. The number of nodes are represented by $L(n) = \log_{\frac{1}{p}} n$, And the maximum of $L(n)$ is:

$$L(n) = 1 - (1 - p^k)^n \le np^k$$

There are $n/2^i$ elements in i-level chain, the probability of new elements belong to i- level chain is $1/p$ when inserted. Therefore, the possibility of the new elements identified as the element of i-level chain is p^i. Chain series is $[\log_{\frac{1}{p}} n] + 1$ for the general p. In this case, there is one element in i-level chain in every $1/p$ of (i-1)-level chain.

The complexity of search, insert and deleting operation are O(n+MaxLevel) while there are n elements in the jump table. In the worst cases, there are only one MaxLevel- level chain element and all the remaining elements are all in level 0 chain. If i>0, the time complexity is O(MaxLevel) in i-level chain while it is O(n) in 0-level chain. The average complexity of each operations of the jump table (search, insert and delete) is O(logn), the proof is as follows:

m as layers, there are two elements skipping the last a elements in each layer, and the time complexity is $T_m = na^{-m} + ma$.

$T_m' = -mna^{-m-1} + m$, declare $T_m' = 0$, then $a = n^{\frac{1}{m+1}}$, $T_m = n^{1-\frac{m}{m+1}} + mn^{\frac{1}{m+1}} = (1+m)n^{\frac{1}{m+1}}$.

The larger value of M, the smaller of a, the minimum value of a is 2. Then $\log_2 2 = \frac{1}{m+1}\log_2 n$, $\frac{1}{m+1} = \log_2 n$, $T_m = 2\log_2 n$ .namely the time complexity is $O(\log n)$.

4.2 Space Complexity

Space complexity, namely all elements could be MaxLevel-level in the worst case and each element needs MaxLevel+1 pointers. For this reason, the n elements need to be stored and meanwhile chain pointer need to be stored also (space needed is O(n * MaxLevel)). Usually, only n*p elements in 1-level chain, $n*p^2$ elements in 2-level chain and $n*p^i$ in i-level chain. So the average of pointer field (not including head to tail node pointer) is $n\sum p^i = n/(1-p)$. Although large space is required in the worst case, but the average space requirements is not large. When $p=0.5$, average space requirement (include the pointers of n nodes) is about 2n pointers' spaces, the proof is as follows:

The first layer n, the second n/2, the third layer $n/2^2$,…, until n/2log n=1. So, the total space requirement is:

$$S = n + n/2 + n/2^2 + ... + n/2\log n < n\,(1 + 1/2 + 1/2^2 + ... + 1/2^{\infty}) = 2n.$$

Therefore, its space complexity is 2n = O (n).

In the distributed environment, the time complexity of jumps map [12] is O (nlogn) and space complexity is O(n). In the same environment, this paper puts forward the algorithm that time complexity is better than that of jumping graph, the space complexity is the same, it is superior to jump graph algorithms overall.

5. Conclusion

This paper analyzes the characteristics and the lack of data positioning mechanism of existing distributed storage system, and puts forward to the weight value jump table distributed data location (DLPSL) strategy, and designs and realizes the distributed storage system. Analysis shows that compared the efficiency of the insert, deleting and positioning of DLPSL with that of single table storage structure and doubly linked list structure not with weights, introducing a weight value can better solve the problem of locating data efficient of distributed storage system, and it has a fast and efficient search efficiency and better application value.

Acknowledgements

The paper supported by the National Natural Science Foundation of China (No.61073196&61272458) and Natural Science Research Foundation of Shanxi Province, China (No. 2011JM8026).

Corresponding Author:

Author Name: Gen-qing Bian
Email:bgq_00@163.com
Mobile Telephone:13319273850
Corresponding Address: School of Information and Control Engineering ,Xi'an University of Architecture and Technology
Postcode: 710055

References

[1]Sikalinda P G, Walters L, Kritzinger P S. A storage system workload analyzer, CS06-02-00[R]. Cape Town, University of Cape Town,2006.
[2]Chanho Ryu,Eunmi Song et al. Hybrid-TH: a Hybrid Access Mechanism for Real-Time Memory-Resident Database Systems[C]// Real-Time Computing System and Applications. Proceeding of the Fifth International Conference. Hiroshima, IEEE .1998:303-310.
[3]LIN X,SHROFF N B. An optimization-based approach for QoS routing in high-bandwidth networks [J]. IEEE/ACM Transactions on Networking, 2006, 14(6): 1348-1361.
[4] Yue Ma , Dayong Zhan,Yicheng Jin. DDM intersection area fast query algorithm Based on jump table [J] The computer simulation magazine,2005 22(7):46—49.
[5] J.Aspnes and G.Shah. Skip graphs[J]. ACM Trans.on Algorithms, 2007，3(4):37—42.
[6] Aspnes, J. ,Wieder, U. The expansion and mixing time of skip graphs with applications[C]//ACM. In Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures. New York, ACM,2005:126–134.
[7] Jun Ying, Binmao Yang.A kind of multiple index of large-scale data rapidly search algorithm [J]. Computer Science magazine, 2009 36(3):258—260.
[8]John Risson ,Tim Moors. Survey of research towards robust peer-to-peer networks: Search methods[J] Computer Networks 2006, 50(17): 3485—3521.
[9] William Pugh.Skip lists: a probabilistic alternative to balanced trees[J]. Communications of the ACM June,1990,33(6):668-676.
[10] W Pugh. Skip lists: a probabilistic alternative to balanced trees [J].Communications of the ACM, 1990, 33(6): 668-676.
[11] Mark Allen Weiss, Data Structures and Algorithm Analysis in C++ (Third Edition)[M]. Boston. Published by Addison-Wesley, 2006:85-126.
[12] Hammurabi Mendes, Cristina G.Fernandes. A Concurrent Implementation of Skip Graphs[J]. Electronic notes in discrete mathematics.2009, 35:263-268.