Advanced Materials Research *Online: 2012-01-03*
ISSN: 1662-8985, Vols. 433-440, pp 5129-5135
doi:10.4028/www.scientific.net/AMR.433-440.5129

Clustering Columns of the Wide-Table in Cloud Computing

Huang Bin[1, a], Peng Yu-Xing[2, b]

[1]School of Computer Science，Wuhan University，Wuhan, China

[2] National Laboratory of Parallel and Distributed Processing, School of Computer Science，National University of Defense Technology，Changsha, China

[a]hb415@163.com, [b]pengyuxing1963@yahoo.com.cn

Keywords: Clustering algorithm, Wide table, Entropy, Partition.

Abstract. Various data-centric web applications are becoming the developing trend of information society. Cloud computing currently adopt column-oriented storage wide table to represent the heterogeneous structured data of these applications. The wide table reduces the waste of storage space, but slows down query efficiency. The paper implements the hybrid partition on access frequent (HPAF) to horizontally and vertically partition a wide table. It uses a variant of consistent hashing to dynamically horizontally partition a wide table across multiple storage nodes on each node's performance; It use entropy to represent the number of reducing access data block from the table with N columns than from N column-oriented storage tables. According to the second law of thermodynamics, the paper designs an entropy increasing clustering algorithm to classify the columns of a wide table. The algorithm finds a cluster with multiple classes which save maximum access time. The paper implements an algorithm for structured query across multiple materialized views too. Lastly the paper demonstrates the query performance and storage efficiency of our strategy compared to single column storage.

Introduction

Various Internet services based on large scale data are becoming the developing trend of information society, and they include [1]: (1) new generation of e-commerce system (e.g., Amazon [2]) have collected the information of various products, which consolidates information about products from thousands of manufacturers and distributors. It requires data schema to constantly evolve that new products joining in the system brings with new attributes to be added to the current category.(2) Data publishing and annotating services based on Web 2.0, such as Delicious [3], Flickr [4], Google Co-op [5]. These services share the same operation mode: users publish data items, such as blog, pictures, advertisements, etc. which is associated with simple descriptions such as tags and attributes created by users themselves. Users can define their own, variable numbers of attributes to describe their published objects. (3) Search engine, such as Google [6]. These services use Web Spider to crawl large scale web pages, then analyze, index and rank them, lastly accept each user's search. Crawling web pages always creates unpredictable attribute-value pairs. (4) Some novel web applications also create such new type of web data. Google Base [7] is an attempt to enable content owners to upload structured data items into Google and can be searched. Users describe the data they upload into Google Base using a self-defined item type and attribute-values pairs. The result is that Google Base is a very large, self-describing, semi-structured, heterogeneous database. All aforementioned applications are data-centric web applications and procedure massive data which characteristic is as follows:

(1) Massive and increasing. Terabyte-scale even petabyte-scale data require the storage system must scale incrementally.
(2) Large Number of Columns. For example, Agrawal [8] point out that in an e-commerce marketplace for the electronics industry the catalog may contain 2000 categories, with 5000 attributes per category. The current database system always has a limit on the numbers of columns in a table. The mast large number of the column in DB2 or oracle is1012.
(3) Schema Evolution. The increasing of heterogeneous data bring table into frequently altering, which is expensive for conventional relation database.
(4) Sparsity. NULLs should occupy most of the fields. It is a waste of storage space. What is more, it increases the index size and affects the sort order of index.

The above new characteristic mentioned make the usual static, row-oriented storage database not support for representing, storing and managing the data of these applications mentioned. In cloud, the current practice to solve this problem is to adopt wide table [1, 9, 10, 11], which stores all data into a logic table. The wide table discards row-oriented storage implementation mechanisms, as they not only introduce large number of nulls which waste storage space due to the wide table has thousands of columns, but also so much columns that multi-dimension structured query unfeasible. The wide table adopts column-oriented storage implementation mechanisms [1, 12, 13, 14, 15] which reduce the waste of storage space, but slow down query due to introduce a lot of join operations for structured query. Therefore, the cloud needs new wide table management techniques to reduce the waste of storage space and improve query efficiency.

We implement the hybrid partition on access frequent (HPAF) to horizontally and vertically partition a wide table. It form all storage nodes into a logic ring, and uses a variant of consistent hashing to dynamically horizontally partition a wide table across multiple storage nodes based on each node's performance. We call all data of each node a data shard. After counting concurrent access frequency between two columns, it vertically partition the data shard of each node by grouping these columns of the high concurrent access frequency into a column class and building the materialized view for each column class. Not accessing whole wide table, a structured query only access a/multiple materialized view(s), and reduces join operation. Therefore HPAF can improve query efficiency. Due to a kind of query is always related to a column set, in this case, materialized view can cut out a lot of nulls. Therefore, HPAF can improve storage efficiency. We use entropy to represent the number of reducing access data block from the table with N columns than from N column-oriented storage tables. According to the second law of thermodynamics, we design an entropy increasing clustering algorithm (EICA) to classify the columns of a wide table. The algorithm can find a cluster which save maximum access time. The article implements an algorithm for structured query across multiple materialized views too.

Related Work

Google's Bigtable [16] may be the most successful real application of Wide Table. It is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes. Its data model is as followed:

(row:string, column:string,time:int64)→string

Bigtable maintains data in lexicographic order by row key. The row range for a table is dynamically partitioned. Each row range is called a tablet, which is the unit of distribution and load balancing. As a rule, the number of distinct column families in a table should be small, and column families rarely change during operation. On the contrary, columns in a column family are unbounded.

In an alternative implementation which is called HBase[17] in Hadoop[18] project, it first employs binary representation[19,20] for each column families, and uses the vertical representation in each column family table.

The common between HPAF and the hybrid partition mechanism of both Bigtable and HBase is that they first horizontally partition a wide table across all storage nodes, and then vertically partition each data shard in each node. The different between them is: column families in Bigtable and HBase rarely change, whereas column families are changed with user access characteristic in HPAF.

The Hybrid Partition on Access Frequent

Horizontal Partition. High increasing of massive data requires storage system scale incrementally. HPAF relies on consistent hashing to dynamically partition the data across multiple storage hosts in the system. In consistent hashing, the output range of a hash function is treated as a fixed circular space or "ring". Each data item identified by a key is assigned to a node by hashing the data item's key to yield its position on the ring, and then walking the ring clockwise to find the first node with a position larger than the item's position.

The principle advantage of consistent hashing is that departure or arrival of a node only affects its immediate neighbors and other nodes remain unaffected. The basic consistent hashing algorithm presents some challenges. First, the random position assignment of each node on the ring leads to non-uniform data and load distribution. Second, the basic algorithm is oblivious to the heterogeneity in the performance of nodes. To address these issues, HPAF uses a variant of consistent hashing [21]: instead of mapping a node to a single point in the circle, each node gets assigned to multiple points in the ring. To this end, HPAF uses the concept of "virtual nodes". A virtual node looks like a single node in the system, but each node can be responsible for more than one virtual node. Effectively, when a new node is added to the system, it is assigned multiple positions in the ring.

Vertical Partition. To achieve high availability and durability, a cloud system replicates its data on multiple hosts, for example, Bigtable has three copies in GFS. If a copy of a wide table is built materialized views which store data by employing interpreted storage format [22] and other copies are stored on column stores, this method can reduced waste storage space for null and achieve good storage efficiency[1,10,22], meanwhile materialized views can promote access efficiency. A cluster is composed of these columns which concurrent access frequency is high, therefore the probability of all or part column in the cluster is high, and if building a materialized views for it in advance, this can reduce the time of table reconstruction on structure query. For example, supposed a wide table which is described in Fig.1， the concurrent access frequency of A, B, D is high, so is to C, F, but the access frequency of E together with others is low, Fig.2 shows two nodes' materialized view.

OID	A	B	C	D	E	F
1	×		×			
2		×		×	×	
3	×		×	×		×
4	×	×		×	×	

Figure 1. A wide table

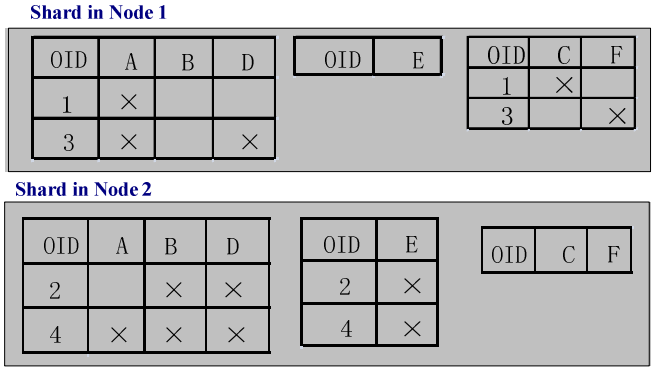


Figure 2. Materialized view in two nodes

The Entropy Increasing Clustering Algorithm

The Cost of Join. Supposed the capacity of a node's cache is C, table Z is composed of N columns which are represented as $A=(a_1, a_2, \ldots, a_n)$, their width are $Wa_1, Wa_2, \ldots, Wa_n$, their data number are $Ra_1, Ra_2, \ldots, Ra_n$. The t_s is the seek time of a disk. The B is the capacity of each disk block. The S is the transmission bandwidth of a disk IO. The t_c is the consuming time computing block data in memory. Hence, we can get two values:

(1) $tc_z(i)$ denotes the time wasted for accessing $i(1 \le i \le N)$ columns/column from table Z. To access i columns of data, It also needs to read all the N columns data, therefore:

$$tc_Z(i) = \frac{\sum_{j=1}^{N} Ra_j Wa_j}{B}\left(t_s + \frac{B}{S} + \frac{t_c}{C}\right) - \frac{\sum_{j=1}^{i} Ra_j Wa_j}{B}\left(t_s + \frac{B}{S} + \frac{t_c}{C}\right)$$

$$tc_Z(i) = \frac{\sum_{j=1}^{N-i} Ra_j Wa_j}{B}\left(t_s + \frac{B}{S} + \frac{t_c}{C}\right) \quad a_j \in A - A_i$$

Due to $t_f \ll t_c + \frac{B}{S}$, we have:

$$tc_Z(i) = \frac{\sum_{j=1}^{N-i} Ra_j Wa_j}{B}\left(t_s + \frac{B}{S}\right) \quad a_j \in A - A_i$$

The $\mathrm{tc_Z(i)} = \frac{\sum_{j=1}^{N-i} Ra_j Wa_j}{B}$ in above formula represents the superabundant data block number as to access *i* columns' data.

(2) $tj_z(2)$ denotes time cost of joining column *j* and column k. Suppose cache size of column *j* is C_1, and that of column *k* is *C-C1*,therefor:

$$\mathrm{tj_z(2)} = \frac{W_{aj}R_{aj}}{B}\left(t_s + \frac{B}{S}\right) + \frac{W_{aj}R_{aj}}{C_1} \times \frac{W_{ak}R_{ak}}{B}\left(t_s + \frac{B}{S}\right) + \frac{W_{aj}Ra_j}{C_1} \times \frac{W_{ak}R_{ak}}{C - C_1} \times t_c + \frac{W_{aj}Ra_j}{C_1} \times \frac{W_{ak}R_{ak}}{C - C_1} \times t_c$$

Due to $t_c \ll t_s + \frac{B}{S}$, we have:

$$\mathrm{tj_z(2)} \approx \frac{\left(t_s + \frac{B}{S}\right)}{B}\left(W_{aj}R_{aj} + \frac{W_{aj}R_{aj}W_{ak}R_{ak}}{C_1}\right)$$

So the *i* columns' join cost $tj_z(i)$ is:

$$\mathrm{tj_z(i)} \approx \left(t_s + \frac{B}{S}\right)\left(\sum_{j=2}^{i} \frac{W_{aj}R_{aj}}{B} + \frac{\sum_{j=2}^{i} R_{aj}W_{aj}\sum_{k=1}^{j-1} R_{ak}W_{ak}}{BC_1}\right)$$

If the i columns are joined together a table, the above formula is the time saved for concurrent accessing i columns, and $\sum_{j=1}^{i} \frac{W_{aj}R_{aj}}{B} + \frac{\sum_{j=1}^{i} R_{aj}W_{aj}\sum_{k=1}^{j-1} R_{ak}W_{ak}}{BC_1}$ is the data block number saved. Therefore, accessing i columns' data from N columns' table, the data block number is:

$$\mathrm{Dz(i)} = \sum_{j=2}^{i} \frac{W_{aj}R_{aj}}{B} + \frac{\sum_{j=2}^{i} R_{aj}W_{aj}\sum_{k=1}^{j-1} R_{ak}W_{ak}}{BC_1} - \frac{\sum_{m=1}^{N-i} R_{am}W_{am}}{B}$$

Definition. *Definition 1: The Association between Attributes and the Association Distance*

If a query accesses simultaneously attribute a_x and a_y, they is associated with each other, marked as $D(a_x, a_y)$. The frequency of concurrent access to attribute a_x and a_y is named as the association distance between attribute a_x and a_y, marked as $D(a_x, a_y)$.

Definition 2: Association graph of attributes

The association graph of attributes is a weight graph which vertices set is some attributes, and each edge's weight is concurrent access frequency of the two vertexes the edge.

Definition 3: Association Class, Class Order and Group

If the association graph of an attribute set is a connected graph, we name the set as an association class, represented as *C*. The number of attributes in the class is called as its order, marked as |*C*|. Multiple disjoint class is termed as group, marked as G.

Definition 4: The Association Distance between Two Attributes

For a wide table with thousands of columns, it is impossible to record the concurrent access probability of any number of attribute combinations. Therefore, we use a simple method which uses a n-by-n matrix to record the access frequency of two attributes, the matrix's row and column is all attributes of the wide table, and n is the number of the attributes. For a query with n attributes ($a_1, a_2, \ldots a_n$), the *M[i][j]* is added *1/t*, *t* is the total access number and *M[i][j]* represented as the association distance between attribute *i* and *j*.

Definition 5: The Association Entropy

For a Column-oriented storage wide table, it need reconstruct a horizontal table by joining multiple attributes as the system executes a multi-key query. The operation are defined as $\mathrm{LOJ}_{i=1}^{n}(B_i)$, LOJ denotes left outer join. It is time-consuming.

Association matrix only maintains the access frequency of two attributes. In this case, we can get the probability of concurrent access *N* attributes *p(Z)*, which satisfy the following relations: $0 \leq \mathrm{p(Z)} \leq \min\{\mathrm{D(x,y)} | x, y \in Z, |Z| = N\}$.

There are C_N^i combinations for access to i attributes from class C which order is N. if each combination can save access time $\Delta t(i)$, the total saving time of combinations is:

$$\Delta t = \sum_{|Z|=1}^{N} \sum_{i=1}^{C_N^{|Z|}} p_{max}(i)\Delta t_z(i)$$

$$p_{max}(i) = \min\{D(x_i, y_i)|x_i, y_i \in Z, Z \sqsubseteq C, |Z| = j, 2 \le j \le N\}$$

$$\Delta t = \frac{\left(t_s + \frac{B}{S}\right)}{BC_1} \sum_{|Z|=1}^{N} \sum_{i=1}^{C_N^{|Z|}} p_{max}(i) D_Z(i)$$

We name $\sum_{|Z|=2}^{N} \sum_{i=1}^{C_N^{|Z|}} p_{max}(i) D_Z(i)$ as the association entropy of class C, marked as $H(C)$, which physically means that when access to i columns, the number of access data block from the table with N columns is less $H(C)$ than from i column-oriented storage tables.

Suppose $|G|=m$, $G=\{C_1, C_2, ..., C_m\}$, the association entropy of cluster G is $(G) = \sum_{i=1}^{m} H(C_i)$.

Definition 6: The Association Entropy between Attribute x and class C

We name increasing entropy adding attribute x to class C as the association entropy between attribute x and class C, marked as H(x,C), it is:

$$H(x, C) = \sum_{|Z|=1}^{N} \sum_{i=1}^{C_N^{|Z|}} p_{max}(i) D_z(i) \qquad |Z| = j, Z \sqsubseteq C,$$

$$1 \le j \le N, \quad p_{max}(i) = \min\{D(x_i, y_i)|x_i, y_i \in \{x\} \cup Z\}$$

Clustering Algorithm. From the definition of association entropy, we can see the average access time is least, while the system's association entropy arrives maximum. Therefore, the clustering algorithm needs to find the cluster which cluster association entropy is most. The second law of thermodynamics states that in a non-equilibrium system isolated from its environment, the total entropy of that system will tend to increase in an irreversible thermodynamic process. According to the law, we design a clustering algorithm to classify the columns of a wide table. The basic thinking of the algorithm is: Initially, each column is a class. Then, step by step, it finds each column which can make the total association entropy of the cluster increasing when it leave a class and join in another class. Lastly, it finish when the entropy cannot increase, in other word, the cluster arrive balance. At this time the cluster's entropy is most. The algorithm is illustrated in Algorithm 1.

Algorithm 1 cluster(Set AttrSet)

```
for(x: AttrSet)
   G←{xi};
while(true){
  for(x: AttrSet)
    for(C:G)
     if (x∈C){
       decreaseH＝H(x,C-x);existClassNum=N(C);
     }
else{
          IncreaseH=H(x,C);
          if(IncreaseH >maxIncreaseH){
          maxIncreaseH= IncreaseH;
          maxClassNum=N(C); maxattr=x;
      }
if(MaxIncreaseH>0){
   G[maxClassNum]＝G[maxClassNum]∪{x};
   G[existClassNum]＝G[existClassNum]-{x};
}
else
   break;
}
```

Algorithm 2 query(Set AttrSet)

```
while(AttrSet is not empty){
    attr=firstAttr(AttrSet);C←findClass(attr)
    AttrSet= AttrSet-C;Group←C;
 }
if(Group.length==0)
     Out Error;
else
   if(Group.length==0)
      datafile= first(group);
   if(Group.length>1){
      datafile =OIDSet;
      for(C:Group){
         firstclass=first(group);

datafile = datafile ⋉ ∏AttrSet ∩firstclass (firstclass)
       }
    }
fulfill query in datafile
```

Query

The column-class map was stored in memory by a lexicographic order of column; hence we can employ binary search method to find these corresponding classes of a column set. On basis of the map, we design a query algorithm which is illustrated in Algorithm 2.

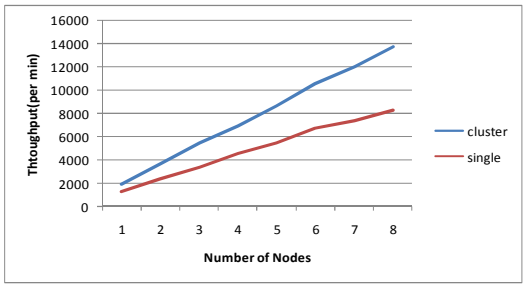
Experiment and Evaluation

Experiment Environment. The experiment environment is located in a LAN and contains six computers with a 2.8GHz Pentium dual-core processor, 2GB memory and 160GB storage. All computers are connected into a subnet by a two-layer switch which ports' speed is 1000Mbps. Our system is implemented in Java 1.6.0. We randomly generated 1000 columns. Table 1 lists the type, column number and width of each column.

Table1 The Status of a Wide-Table

Column Type	Column Width	Tuple Number
char	2	103
long	8	131
float	4	110
double	8	112
string	50	116
string	100	93
string	150	99
string	200	133
string	250	103

The Performance of Query. We choose randomly several columns when we insert a record into the wide table every time. All data are uniformly distributed into the eight nodes and each node has 480k records. We generate exact queries and range mult-keys queries for the keys in zipfian distribution. Without any index, the same queries are fulfilled in the column-oriented storage table and cluster-oriented storage table individually. Their throughout are showed in Figure 3.



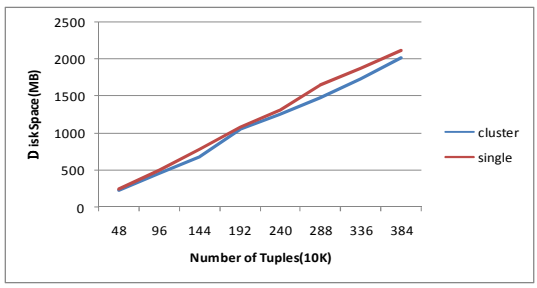


Figure 3. Throughout of query (per minute)

Figure 4.The Storage efficiency

The Storage Efficiency. Each column file employ interpreted storage, so do each class. Figure 4 shows the disk space occupied the tuples varied from 480k to 3840k in column-oriented storage table and cluster-oriented storage table.

Conclusion

The entropy increasing clustering algorithm can classify the columns of a wide table and finds a cluster which save maximum access time. It can in general greatly save access time of structured query by building materialized view for the cluster. Adopting interpreted attribute storage format, the waste storage space of null is few. HPAF uses a variant of consistent hashing to dynamically horizontally partition a wide table across multiple storage nodes on each node's performance. It make storage system can scale incrementally and meet high increasing of massive data.

The author's work is supported by National Basic Research Program of China (973 Program) under Grant No.2011CB30260 and National Natural Science Foundation of China (NSFC) under Grant No.60903205.

References

[1] Bin YANG, Weining QIAN, Aoying ZHOU, Using Wide Table to manage web data: a survey, Front. Comput. Sci. China 2008, 2(3): 211–223.

[2] http://www.amazon.com/

[3] Delicious website. http://www.delicious.com

[4] Flickr website. http://www.flickr.com.

[5] Google co-op website. http://www.google.com/coop

[6] www.google.com

[7] Google base website. http://base.google.com.

[8] Agrawal R, Somani A, Xu Y. Storage and querying of e-commerce data. In: Proceedings of the 27th International Conference on Very Large Data Bases, 2001, 149–158.

[9] Chu E, Beckmann J, Naughton J. The case for a wide-table approach to manage sparse relational data sets. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, 2007, 821–832.

[10] Bei Yu, Guoliang Li, Beng Chin Ooi, Li-Zhu Zhou. One Table Stores All: Enabling Painless Free-and-Easy Data Publishing and Sharing.

[11] Abadi d j. Column stores for wide and sparse data. In:Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR), 2007.

[12] Stonebraker M, O'Neil E, O'Neil P, et al. C-store: a columnoriented DBMS. In: Proceedings of the 31st International Conference on Very Large Data Bases, 2005, 553–564.

[13] Hoque A S M L. Storage and querying of high dimensional sparsely populated data in compressed representation. In: Proceedings of the First EurAsian Conference on Information and Communication Technology, 2002, 418–425

[14] Boncz P, Zukowski M, Nes N. MonetDB/X100: hyper-pipelining query execution. In: Proceedings of the Second Biennial Conference on Innovative Data Systems Research (CIDR),2005

[15] Chang F, Dean J, Ghemawat S, et al. Bigtable: a distributed storage system for structured data. In: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI06), 2006, 205–218

[16] Hbase website. http://wiki.apache.org/lucene-hadoop/Hbase

[17] Hadoop website. http://lucene.apache.org/hadoop

[18] Copeland G P, Khoshafian S N. A decomposition storage model. ACM SIGMOD Record, 1985, 14(4):268–279

[19] Khoshafian S, Copeland G P, Jagodis T, et al. A query processing strategy for the decomposed storage model. In ICDE, 1987, 636–643.

[20] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. SIGOPS, 2007

[21] J. L. Beckmann, A. Halverson, R. Krishnamurthy, and J. F.Naughton. Extending RDBMSs to support sparse datasets using an interpreted attribute storage format. In Proc. of ICDE,2006.