H-HIBASE: Compression Enhancement of HIBASE Technique Using Huffman Coding

Ahsan Habib
Metropolitan University, Sylhet, Bangladesh
Email: ahabib@metrouni.edu.bd

A. S. M. Latiful Hoque
Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
Email: asmlatifulhoque@cse.buet.ac.bd

Md. Russel Hussain
Metropolitan University, Sylhet, Bangladesh
Email: mrhussain@rocketmail.com

***Abstract*— HIBASE compression technique simply replaces the attribute values in a tuple with fixed length code-words. However, fixed length coding system is not an optimal compression technique because some redundancies occur in the compressed table. This redundancy can be avoided if we use Huffman code-words. Moreover, using Huffman code-word will ensure optimal compression as well as high performance operation. The objectives of the research are to i) develop a dictionary by applying the principle of Huffman coding, ii) compress the relational storage of HIBASE by applying dynamic Huffman coding, iii) develop algorithm to perform query operation on the compressed storage, iv) analyze the performance of the proposed system in terms of both storage and queries. The main contribution of this research is to develop a compression technique. It implies the enhancement of HIBASE technique using HUFFMAN coding (H-HIBASE) with better compression capability.**

***Index Terms*— data compression, database compression, HIBASE, Huffman, variable length coding.**

I. Introduction

The HIBASE (High Compression Database System) [1] approach is a compression technique for Main Memory Database Management System (MMDBMS) [2] which supports high performance query operations on relational structure [3]. The dictionary space overhead is excessive for this system. Fixed length coding system does not consider the frequency of occurrence of the values. Thus HIBASE requires higher space in the compressed database. This higher storage requirement can be avoided if we use Huffman code-words [4]. As we know Huffman algorithm generates an optimal tree [4][5], hence the compression will be optimized. However, the use of Huffman coding could increase the query complexity in HIBASE, but this complexity can be reduced by designing proper algorithm.

A. Dictionary Based HIBASE Compression Approach

The HIBASE [1] approach is a more radical attempt to model the data representation which is supported by information theory. The architecture represents a relation table in storage as a set of columns, not as a set of rows. Of course, the user is free to regard the table as a set of rows. However, the operation of the database can be made considerably more efficient when the storage allocation is arranged by columns.

TABLE I.

DISTRIBUTOR RELATION

ID	First Name	Last Name	Area
1	Abdul	Bari	Dhaka
2	Abdur	Rahman	Sylhet
3	Md	Alamin	Chittagong
4	Abdul	Gafur	Dhaka
5	Salam	Bari	Sylhet
6	Md	Tuhin	Rajshahi
7	Salam	Mia	Rajshahi
8	Chan	Mia	Dhaka
9	Ghendhu	Mia	Chittagong
10	Abdur	Rahman	Sylhet

The database is a set of relations. A relation is a set of tuples. A tuple in a relation represents a relationship among a set of values. The corresponding values of each tuple belong to a domain for which there is a set of permitted values. If the domains are D_1, D_2,……., D_n respectively, a relation r is defined as a subset of the Cartesian product of the domains. Thus r is defined as $r \subseteq D_1 \times D_2 \times .....\times D_n$.

An example of a relation is given in Table I. In the conventional database technology, we have to allocate enough space to fit the largest value of each field of the records. When the database designer does not know

doi:10.4304/jcp.8.5.1175-1183

exactly how large the individual value is, he/she must make a mistake on the side of caution and make the field larger than is strictly necessary. In this instance, a designer should specify the width in bytes as shown in Table II. Each tuple is occupying 18 bytes, so that 10 tuples occupy 180 bytes.

TABLE II.

FIELD LENGTH AND TUPLE SIZE FOR DISTRIBUTOR RELATION

# of Attribute	Attribute Name	Bytes
0	First Name	6
1	Last Name	6
3	Area	6
	Total	**18**

The HIBASE architecture can be derived from a conventional record structure using the following steps:

1. A dictionary per domain is employed to store the string values and to provide integer identifiers for them. This achieves a lower range of identifier, and hence a more compact representation could be achieved.
2. Replace the original field value of the relation by identifiers. The range of the identifiers is sufficient to distinguish string of the domain dictionary.

TABLE III.

COMPRESSED TABLE IN HIBASE

First Name	Last Name	Area
000	001	00
010	010	01
011	011	10
000	100	00
001	001	01
011	101	11
001	000	11
100	000	00
101	000	10
010	010	01

Hence in the compressed table each tuple resumes only 3 bits for First Name, 3 bits for Last Name, 2 bits for Area forming total of 8 bits. This is not the overall storage; however, we must take account of the space occupied by the domain dictionaries and indexes. Typically, a proportion of domain is present in several relations and this reduces the dictionary overhead by sharing it through different attributes.

B. The Huffman Codes

In computer science and information theory, Huffman coding is an entropy encoding algorithm used for lossless data compression. The term refers to the use of a variable-length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol. It was developed by David A. Huffman while he was a Ph.D. student at MIT, published in 1952 paper "A Method for the Construction of Minimum - Redundancy Codes" [4][6].

Huffman coding is a widely used and very effective technique for compressing data;

1. Savings of 20% to 90% are typical, depending on the characteristics of the file being compressed.
2. Huffman coding involves the use of variable-length codes to compress long string of text.
3. By assigning shorter codes to more frequent characters, Huffman encoding can compression text by as much as 80%.

The simplest construction algorithm uses a priority queue where the node with lowest probability is given highest priority [6]:

1. Create a leaf node for each symbol and add it to the priority queue.
2. While there is more than one node in the queue:
   a) Remove the two nodes of highest priority (lowest probability) from the queue
   b) Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.
   c) Add the new node to the queue.
3. The remaining node is the root node and the tree is complete.

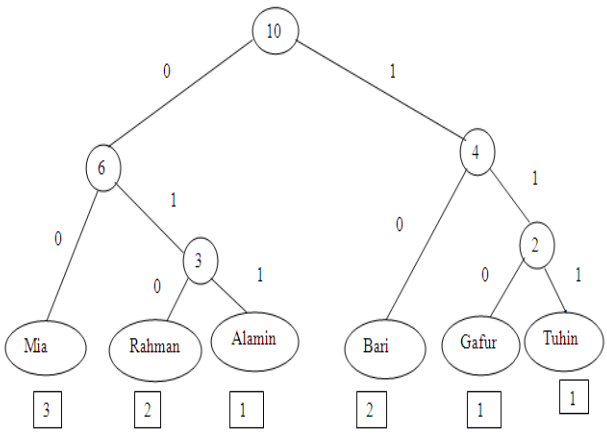

Figure 1. Construction of a Huffman Tree.

Since efficient priority queue data structures require O(log n) time per insertion, and a tree with n leaves has $2n-1$ nodes, this algorithm operates in O(n log n) time.

II. EXPERIMENTAL DESIGN

As we know Huffman algorithm generates an optimal tree, hence the compression will be optimized. Figure 2 shows the whole analysis at a glance, five steps are necessary to complete whole process. Steps are explained below:

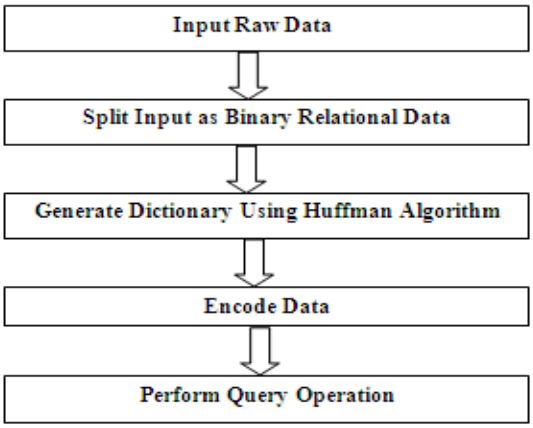

Figure 2. Experimental Design at a glance.

Moreover high performance will be ensured as most repeated attribute values will get more weight and will be entered first in the dictionary i.e. domain dictionary values will be sorted in such a way that frequently occurred values will be accessed first than the rare values. It has been shown in figure 2 that five steps are necessary to complete the whole process.

A. *Five Steps of the H-HIBASE Architecture*

Step 1: *Take synthetic and real data as input (Consider the database shown in table IV).*

TABLE IV.
DISTRIBUTOR RELATION

ID	First Name	Last Name	Area
1	Abdul	Bari	Dhaka
2	Abdur	Rahman	Sylhet
3	Md	Alamin	Chittagong
4	Abdul	Gafur	Dhaka
5	Salam	Bari	Sylhet
6	Md	Tuhin	Rajshahi
7	Salam	Mia	Rajshahi
8	Chan	Mia	Dhaka
9	Ghendhu	Mia	Chittagong
10	Abdur	Rahman	Sylhet

Step 2: *Split the relational database as binary relational databases (Shown in table V).*

Binary relational database is a database with two columns in each table and it is very efficient where column wise searching is regular. The Table IV has been split to three Binary relation tables which are shown in Table V.

Step 3: *Generate dictionary using Huffman algorithm (Shown in table VI).*

A dictionary for each domain is created and it stores string values and provides Huffman codeword for them. This achieves a lower range of codeword, and hence a more compact representation can be achieved.

TABLE V.
BINARY RELATIONAL DATABASE

ID	First Name
1	Abdul
2	Abdur
3	Md
4	Abdul
5	Salam
6	Md
7	Salam
8	Chan
9	Ghendhu
10	Abdur

ID	Last Name
1	Bari
2	Rahman
3	Alamin
4	Gafur
5	Bari
6	Tuhin
7	Mia
8	Mia
9	Mia
10	Rahman

ID	Area
1	Dhaka
2	Sylhet
3	Chittagong
4	Dhaka
5	Sylhet
6	Rajshahi
7	Rajshahi
8	Dhaka
9	Chittagong
10	Sylhet

TABLE VI.
H-HIBASE DICTIONARY

Index	First Name	Codeword
1, 4	Abdul	110
5, 7	Salam	111
2, 10	Abdur	00
3,6	Md	01
8	Chan	100
9	Ghendhu	101

Index	Last Name	Codeword
7,8,9	Mia	10
1,5	Bari	111
2, 10	Rahman	00
3	Alamin	010
4	Gafur	011
6	Tuhin	110

Index	Area	Codeword
1,4,8	Dhaka	10
2,5,10	Sylhet	11
3,9	Chittagong	00
6,7	Rajshahi	01

Step 4: *Develop algorithm to encode data (Shown in table VII).*

The range of the identifiers needs only to be sufficient to distinguish unambiguously which string of the domain dictionary is indicated. In Table III, since there are only 6 distinct First Names, only six variable length codeword are required. This range can be represented by only a 3 bit or 2 bit binary number.

TABLE VII.
H-HIBASE STORAGE

First Name	Last Name	Area
110	111	10
00	00	11
01	010	00
110	011	10
111	111	11
01	110	01
111	10	01
100	10	10
101	10	00
00	00	11

Therefore in the compressed table each tuple requires only (2 bits or 3 bits for First Name, 2 bits or 3 bits for Last Name, 2 bits for Area) a total of maximum 8 bits. This achieves a compression of the table by a factor of

over 15.

Step 5: *Develop algorithm to perform query operation on the compressed storage.*

After encoding data it is challenging to retrieve those codes from compressed storage. Dictionary access and compressed storage access are necessary to perform every query.

B. H-HIBASE: Storage Complexity

B.1. HIBASE

$SCi = n * C_i$ bits

Where SCi = space needed to store column i in compressed form

n = number of records in the relation

Ci = number of bits needed to represent i^{th} attribute in compressed form

= [lg(m)] ; m is number of entries in the corresponding domain dictionary

Total space to store all compressed columns, $S_{HIBASE} = \sum_{i=1}^{p} S_{ci}$ bits; number of column is p

If we assume that domain dictionaries will occupy an additional 25% of S = 1.25 S, then total space to store the compressed relation, $S_{CRHIBASE} = 1.25\ S_{HIBASE}$

B.2. H-HIBASE

$$S_{H\text{-}HIBASE} = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} \text{ bits}$$

a_{ij} represents the number of bits in a particular position of the two dimensional matrix, where i is the number of row and j is the number of column.

If we assume that domain dictionaries will occupy an additional 25% of S = 1.25 S, then total space to store the compressed relation, $S_{CRH\text{-}HIBASE} = 1.25\ S_{H\text{-}HIBASE}$

B.3. Compression Enhancement

Compression Enhancement = $((S_{CRHIBASE} - S_{CRH\text{-}HIBASE})*100 / S_{CRHIBASE})$ %

III. IMPLEMENTATION

A. H-HIBASE Dictionary

To translate to and from the compressed form it is necessary to go through a dictionary. A dictionary is a list of values that occur in the domain. Huffman dictionary will be comparable to Huffman table where two pieces of information will be stored namely lexeme and token. Lexeme corresponds to discrete values in a domain whereas token corresponds to code-word. Short code-words will be placed first for a domain dictionary which will ensure faster dictionary access. Hence there will be a significant improvement in database performance during compression, decompression and query operations. As Huffman coding gives more weight to most repeated value, it is likely to have shortest code-word to most repeated value. Huffman algorithm will generate the position of values in the dictionary. Table VI shows dictionaries for distributor relationship. The Huffman dictionary has generated as per following algorithm.

```
HUFFMAN (C)
1.   n ← |C|
2.   Q ← C
3.   for i ← 1 to n -1
4.        do allocate a new node z
5.           left[z]  ← x ← EXTRACT-MIN (Q)
6.           right[z] ← y ← EXTRACT-MIN (Q)
7.           f[z] ← f[x] + f[y]
8.           INSERT (Q, z)
9.   return EXTRACT-MIN (Q)
```

In the pseudocode that follows, we assume that C is a set of n strings and each string c ∈ C is an object with a defined frequency f[c]. The algorithm builds the tree T corresponding to the optimal code in a bottom-up manner. It begins with a set of |C| leaves and performs a sequence of |C| - 1 "meaning" operations to create the final tree. A min-priority queue Q, keyed on f, is used to identify the two least-frequent objects to merge together. The result of the merger of two objects is a new object whose frequency is the sum of the frequencies of the two objects that were merged [7].

In algorithm n is the initial queue size, line 2 initializes the min-priority queue Q with the character in C. The **for** loop in line 3-8 repeatedly extracts the two nodes x and y of lowest frequency from the queue, and replaces them in the queue with a new node z representing their merger. The frequency of z is computed as the sum of the frequencies of x and y in line 7. The node z has x as its left child and y as its right child. After n-1 mergers, the node left in the queue-the root of the code tree returned in line 9.

The **for** loop in lines 3-8 is executed exactly n-1 times, and since each heap operation requires time O (lg n), the loop contributes O (n lg n) to the running time. Thus, the total running time of HUFFMAN on a set of n characters is O (n lg n).

B. H-HIBASE: Encoding

Consider a set of source symbols $S = \{ s_0, s_1, \ldots, s_{n-1}\}$= {Dhaka, Sylhet, Chittagong, Rajshahi} with frequencies $W = \{ w_0, w_1, \ldots, w_{n-1}\}$ for $w_0>=w_1>=\ldots>=w_{n-1}$, where the symbol s_i has frequency w_i. Using the Huffman algorithm to construct the Huffman tree T, the codeword c_i, $0<=i<=n-1$, for symbol s_i can then be determined by traversing the path from the root to the left node associated with the symbol s_i, where the left branch is corresponding to '0' and the right branch is corresponding to '1'. Let the level of the root be zero and the level of the other node is equal to summing up its parents level and one. Codeword length l_i for s_i can be known as the level of s_i. Then the wighted external path length $\sum w_i l_i$ is minimum. For example, the Huffman tree corresponding to the source symbols $\{ s_0, s_1, \ldots\ldots,s_7\}$ with the frequencies {3, 3, 2, 2} is shown

in the Figure1. The codeword set C{c_0,c_1,…….,c_7} is derived as {10, 11, 00, 01}. In addition, the codeword set compose of a space with 2^d addresses, where d=2 is the depth of the Huffman tree.

In the following, the detailed algorithm to generate the intervals is presented. For each Huffman tree, the required storage for the interval representation is n entries. Each entry contains two fields: address and symbol. The length of address is d bits, and the storage complexity is O (n).

Both C and C++ allow integer members to be stored into memory spaces smaller than the compiler would ordinarily allow [8]. These space-saving structure members are called *bit fields*, and their width in bits can be explicitly declared. The following structure has three bit-field members: kingdom, phylum, and genus, occupying 2, 6, and 12 bits respectively.

```
struct taxonomy {
        unsigned kingdom: 2;
        unsigned phylum: 4;
        unsigned genus: 12;
                    };
```

To store codeword we have declared an array of a structure with bit field where data can be stored with 1 bit storage. This structure will have 32 members variable named a,b,c,…..,z,A,B,…,F and every member can store 1 bit. To put databits in this structure we have a function named **putvalue** (index_of_structure, data_variable, databit) which will store bit into the structure after reading the input from the dictionary.

```
ENCODE (Huffman_Dictionary hd)
1.  Input: Huffman_Dictionary ( index, name, databit,
    frequency)
2.  Output: Encoded Bit Stream
3.  BEGIN
4.  for i ←1 to total_number_of_rows
5.      for j ← 0 to codeword [i].lenght
6.          putvalue (index_of_structure, data_variable,
            databit)
7.      if (data_variable == 'z')
8.          data_variable ← 'A'
9.      else if (data_variable == 'F')
10.         data_variable ← 'a'; index_of_structure ++
11.     else data_variable ++
12. END
```

The required storage for the interval representation is n entries and the storage complexity is O (n).

IV. Performance Analysis

The objective of the experimental work is to verify the applicability and feasibility of the proposed H-HIBASE architecture. The experimental evaluation has been performed with synthetic and real data. The experimental result is compared with DHIBASE and widely used Oracle 10g. Our target was to handle relations and justify the storage requirements and query time in comparison with DHIBASE and Oracle 10g.

A. Experimental Environment

H-HIBASE has been tested on a machine with 1.73 GHz Pentium IV processor and 1 GB of RAM, running on Microsoft Windows XP. We have created five different relations for synthetic data which is given below. Each query has been executed five times and the average execution time has been taken.

A.1. Data Set For Synthetic Data

A random data generator has been used to generate synthetic data and large numbers of records have been inserted into each table. Five tables in synthetic data set are given below, where first attribute of each table is the primary key. Our synthetic data generator has generated 21035, 21120, 21214, 31422, 30455 records for *Distributor, Customer, Item, Employee* and *Store* relations respectively for 1 MB space in Oracle 10g. It has also been observed that the data generator has generated 24135, 24869, 24567, 362462, 36826 records for *Distributor, Customer, Item, Employee* and *Store* relations respectively for 2 MB space in Oracle 10g. Table VIII shows overall compression rate for different number of record in different relation. Compression rate is calculated with respect to DHIBASE and Oracle 10g. Table IX shows overall CF's for different relations, CF's are calculated with respect to Oracle 10g. We observe that the proposed system outperforms Oracle 10g by a factor of 11 to 13. Table X also shows that the H-HIBASE has greater compression capability than DHIBASE which is between 9% to 15%.

Distributor (d_id, fname, lname, area)
Customer (c_id, name, street, city)
Item (i_id, type, description)
Employee (e_id, name, department)
Store (s_id, location, type)

A.2. Data Set For Real Data

Billing Management Software for managing different types of bills like water bill, electricity bill for super market, different types of report like daily bill, monthly bill, yearly bill can be produced by this software. Real data set of this software has already been shown below where ten tables are available with the following data. Storage requirement in different systems are shown in table XI. Table XII also shows that the H-HIBASE has greater compression capability than DHIBASE which is more than 30%.

Electricbill (issue_no, meterno, presentreading, surcharge, shop_id, tannent_id, bill_month, unit_rate, issue_date, paid_date, demand_charge, meter_charge, last_date, ref_no, consume, vat, previousreading)

Electricbill_for_shop (meter_no, meter_rgd, prv_surcharge, prv_vat, prv_demand, prv_from_date, prv_to_date, shop_id, tannent_id, paid_date, last_date, prv_metercharge, unit_rate, prv_consume, issue_date, is_due, max_rgd, ref_no, prv_arrear, meter_charge, demand_charge)

Floor (floor_id, floor_name)

Rate (rate_id, rate_title, charge)

Shop (shop_number, shop_name, shop_rent, shop_floor_number, shop_id, tannent_id)

Tennant (tannent_id, tannent_name, account_no, address, phone)

Utility_bill (issue_no, bill_month, last_date, paid_date, shop_id, tannent_id, issue_date, ref_no)

Utility_bill_detail (issue_no, utility_id, bill_amt, surcharge)

Utility_bill_for_shop (utility_id, default_amount, prv_amount, prv_surcharge, prv_from_date, prv_to_date, shop_id, tannent_id, last_date, paid_date, is_due, ref_no)

Utility_setup (utility_id, utility_title, default_bill)

A.3. Data Generation Algorithm

From the above algorithm it has been observed that the random data generation function has generated an amount of random data for a column, which has been inserted into the database table.

B. Storage Requirement

B.1. Synthetic Data

```
1    InsertRandomData (RowCount)
2     BEGIN
       LOOP
4       COL1 ← VAR1 ←   dbms_randon.string('L', 10)
5       COL2 ← VAR2 ←   dbms_randon.string('L', 10)
              .
              .
8      COLN ← VARN ←   dbms_randon.string('L', 10)
9           LOOP
10            IF mod(counter,50)=0 THEN
11                 REPEAT step 4 to 8
12            END IF
13            InsertData (COL1, COL2, ...,COLN)
14            Counter ← counter+1
15            Exit when counter>=rowcount
16          END LOOP
17       END LOOP
18     END
```

TABLE VIII.
COMPRESSION ACHIEVED IN KB

Relation	# of Record	Oracle 10g.	DHIBASE	H-HIBASE
Distributor	21035	1024	96.28	81.84
Customer	21120	1024	96.65	82.17
Item	21214	1024	97.10	82.54
Employee	31422	1024	95.88	86.30
Store	30455	1024	92.93	83.64

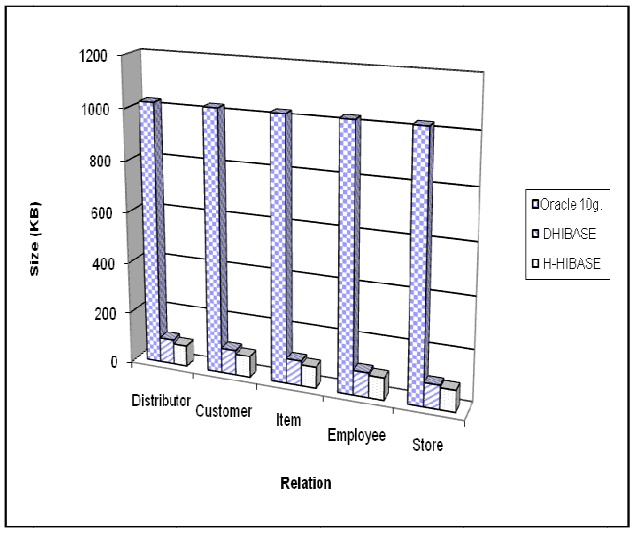

Figure 3. Storage in H-HIBASE, DHIBASE and Oracle 10g.

Figure 3 shows the storage comparison among H-HIBASE, DHIBASE and Oracle 10g. In the figure it has been indicated that DHIBASE can compress the oracle storage with the rate of 90%, whereas H-HIBASE can compress the oracle storage with the rate of 92%.

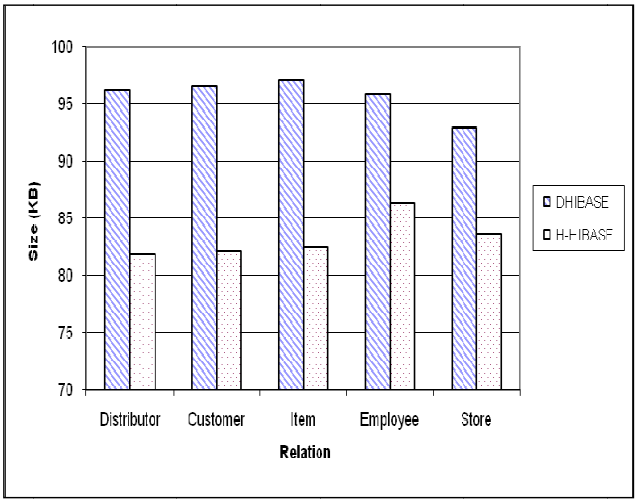

Figure 4. Storage in DHIBASE and H-HIBASE.

Figure 4 indicates the storage comparison between H-HIBASE and DHIBASE. In the figure it has been indicated that H-HIBASE has better compression capability with the rate 30%. This is because DHIBASE has used fixed length coding whereas H-HIBASE has used variable length Huffman coding which needs a reduced amount of storage.

TABLE IX.
COMPRESSION FACTOR (CF) WITH RESPECT TO ORACLE 10G

Relation	# of Record	Oracle 10g. (KB)	H-HIBASE (KB)	Overall CF
Distributor	21035	1024	81.84	12.51
Customer	21120	1024	82.17	12.47
Item	21214	1024	82.54	12.40
Employee	31422	1024	86.30	11.86
Store	30455	1024	83.64	12.24

TABLE X.
COMPRESSION IMPROVEMENT WITH RESPECT TO DHIBASE

Relation	# of Record	DHIBASE (KB)	H-HIBASE (KB)	Enhancement Rate (%)
Distributor	21035	96.28	81.84	14.99
Customer	21120	96.65	82.17	14.98
Item	21214	97.10	82.54	14.99
Employee	31422	95.88	86.30	10.02
Store	30455	92.93	83.64	9.99

TABLE XI.
CODE SIZE IN DHIBASE AND HIBASE FOR SAME NUMBER OF RECORDS

Relation	# of Record	DHIBASE (KB)	H-HIBASE (KB)	Enhancement Rate (%)
Distributor	21035	72.21	61.38	14.99
Customer	21120	72.51	61.63	15
Item	21214	72.83	61.91	14.99
Employee	31422	71.91	64.73	9.98
Store	30455	69.7	62.73	10

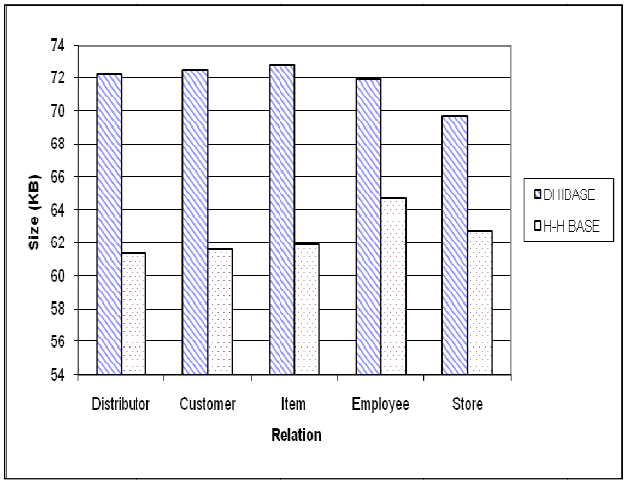

Figure 5. Code size in DHIBASE and H-HIBASE.

The storage of code size in DHIBASE and H-HIBASE technique is shown in Figure 5. From the figure, H-HIBASE produces minimum number of code to store entire relation than that of DHIBASE system. DHIBASE has needed around 70 KB to store code size, whereas H-HIBASE has needed around 60 KB to store code size for the same number of records.

B.1. Real Data

Storage requirement in different techniques for real data has been shown in table XII. It has been observed that the proposed system is better than Oracle 10g by a factor of 4 to 5. Table XII also shows that the H-HIBASE has greater compression capability than DHIBASE which is more than 30%. Higher storage requirement has been avoided by using Huffman code-words in H-HIBASE technique. Moreover high performance has been ensured as most repeated attribute values get more weight and enter first in the dictionary. Domain dictionary values are sorted in such a way that frequently occurred values are accessed first than the rare values.

TABLE XII.
COMPRESSION ACHIEVED IN KB

Relation	# of Col-umn	# of Record	Oracle 10g.	DHIBASE	H-HIBASE
Electric bill	17	1505019	381424	126347	88152
Electric bill_for_shop	17	570			
Floor	3	8			
Rate	3	8			
Shop	6	583			
Tennant	6	292			
Utility_bill	8	1550987			
Utility_bill_detail	4	6206961			
Utility_bill_for_shop	13	2266			
Utility_setup	3	6			

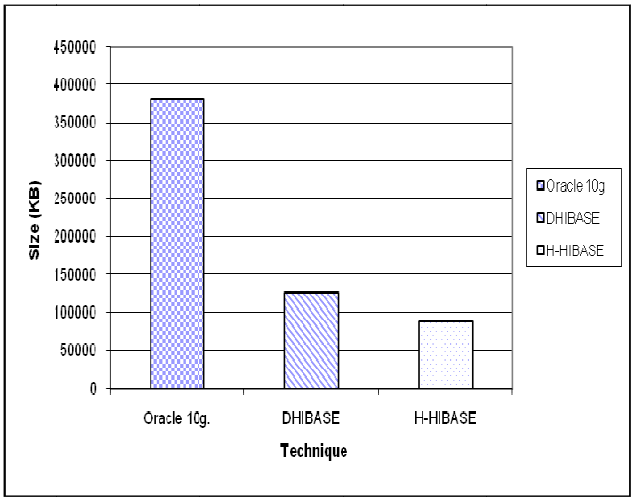

Figure 6. Storage of real data in H-HIBASE, DHIBASE and Oracle 10g.

Figure 6 shows the comparison of storage size among Oracle database, DHIBASE, and H-HIBASE. To store same number of record it is required approximately 380 MB, 125 MB, and 85 MB in Oracle 10g, DHIBASE, and H-HIBASE respectively. H-HIBASE technique has more compression capability than any other existing systems.

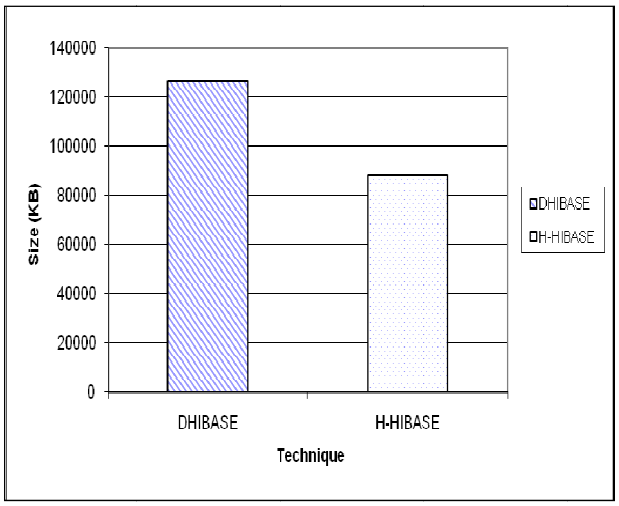

Figure 7. Storage of real data in DHIBASE and H-HIBASE.

Figure 7 indicates the storage comparison between H-HIBASE and DHIBASE. In this figure H-HIBASE has better compression capability with the rate of more than 30%.

TABLE XIII.
COMPRESSION ENHANCEMENT WITH RESPECT TO DHIBASE

Relation	Oracle 10g. (KB)	DHIBASE (KB)	H-HIBASE (KB)	Enhancement Rate (%)
Real Data	381424	126347.42	88152.89	30.23

TABLE XIV.
CODE SIZE COMPARISON

Relation	DHIBASE (KB)	H-HIBASE (KB)	Enhancement Rate (%)
Real Data	31586.75	22038	30.23

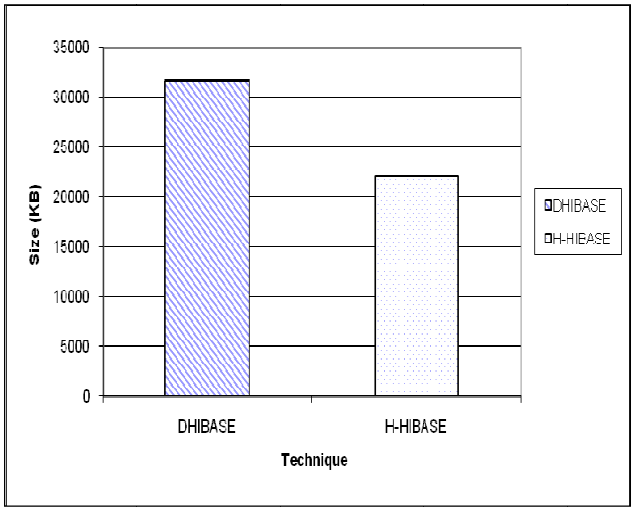

Figure 8. Code size in DHIBASE and H-HIBASE.

Figure 8 shows the code size comparison between H-HIBASE and DHIBASE. H-HIBASE is space efficient, this is because DHIBASE has used fixed length coding whereas H-HIBASE has used variable length Huffman coding. Variable length coding required smaller amount storage than fixed length coding.

V. CONCLUSION

In this paper we have presented the experimental evaluation of the H-HIBASE architecture. We here evaluated the storage performance in comparison with DHIBASE and Oracle 10g. The storage performance achieved in H-HIBASE is 25 to 40 percent better than the Oracle 10g for real and synthetic data. It has also been shown that the storage performance which is achieved in H-HIBASE is 10 to 35 percent better than the DHIBASE for real and synthetic data.

REFERENCES

[1] W. P. Cockshott, D. McGregor, and J. Wilson, "High-Performance Operation Using a Compressed Database Architecture", *The Computer Journal*, Vol. 41, No. 5, pp. 285-296, 1998.

[2] T. J. Lehman, M. J. Carey, "A Study of Index Structures for Main Memory Database Management Systems", *Proceedings of the Twelfth International Conference on Very Large Databases*, pp. 294-303, August 1986.

[3] M. M. Bhuiyan, A. S. M. Latiful Hoque, "High Performance SQL Queries on Compressed Relational Database", *Journal of Computers*, Vol. 4, No. 12, pp 1263-1274, December 2009.

[4] D. A. Huffman, "A Method for the Construction of Minimum Redundancy Codes", *Proceedings of the I.R.E.*, vol. 40, pp. 1098-1101, September 1952.

[5] X. Kavousianos, E. Kalligeros and D. Nikolos, "Multilevel Huffman Coding: An Efficient Test-Data Compression Method for IP Cores", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 26, No. 6, pp 1070-1083, June 2007.

[6] Huffman Coding, Wikimedia Foundation, Inc., Last accessed on 8 August 2012. Last accessed at: http://en.wikipedia.org/wiki/Huffman_coding

[7] T. H. Coreman, C. E. Leiserson, R. L. Rivest, C. Stein, "*Introduction to Algorithms*", second edition, page 387-388, 2002.

[8] Declaring and Using Bit Fields in Structures, IBM, Last accessed on 18 August, 2012. Last Accessed at: http://publib.boulder.ibm.com/infocenter/macxhelp/v6v81/index.jsp?topic=%2Fcom.ibm.vacpp6m.doc%2Flanguage%2Fref%2Fclrc03defbitf.htm

Ahsan Habib received his B.Sc. Engineering in Computer Science and Engineering from Shah Jalal University of Science and Technology (SUST), Sylhet, Bangladesh in 2004, and completed his M.Sc. in Information and Communication Technology in 2012 from Bangladesh University of Engineering & Technology (BUET), Bangladesh. He is currently working as an Assistant Professor of Metropolitan University, Sylhet, in Computer Science & Engineering Department. His research interest includes Data Management, Compression in Database Systems, E-Governance, E-Learning and M-Learning.

Dr. Abu Sayed Md. Latiful Hoque received his PhD in the field of Computer & Information Science from University of Strathclyde, Glasgow, UK in 2003 with Commonwealth Academic Staff Award. He obtained M.Sc. in Computer Science & Engineering and B.Sc. in Electrical & Electronic Engineering from Bangladesh University of Engineering &

Technology (BUET) in 1997 and 1986 respectively. He has been working as a faculty member in the Department of Computer Science & Engineering of BUET since 1990 and currently his position is a Professor. He is a Fellow of Institute of Engineers Bangladesh (IEB) and Bangladesh Computer Society. His research interest includes Data Warehouse, Data Mining, Information Retrieval and Compression in Database Systems.

Md. Russel Hussain is an undergraduate student of Electrical and Electronics Engineering Department of Metropolitan University, Sylhet, Bangladesh.