RLH: Bitmap Compression Technique Based on Run-Length and Huffman Encoding

Michał Stabno
Poznań University of Technology, Institute of Computing Science, Poland
QXL Poland
Michal.Stabno@allegro.pl

Robert Wrembel
Poznań University of Technology, Institute of Computing Science, Poland
Robert.Wrembel@cs.put.poznan.pl
Presentation Outline

- Bitmap index - concept and main characteristics
- Reducing size of bitmap index
- Run Length Huffman compression algorithm
- Run Length Huffman experimental evaluation
- Summary
**Bitmap Index**

- Composed of bitmaps
- A bitmap is a vector of bits
  - every value from a domain has its own bitmap
  - the number of bits = the number of records
  - a given bit corresponds to a given record

- Basic characteristics
  - Efficient in answering equality and range queries
  - BI size depends on the cardinality of an indexed attribute
    - large for high cardinality attributes

<table>
<thead>
<tr>
<th>ID</th>
<th>sex</th>
<th>bitmap_index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>male</td>
<td>0 1</td>
</tr>
<tr>
<td>2</td>
<td>female</td>
<td>1 0</td>
</tr>
<tr>
<td>3</td>
<td>female</td>
<td>1 0</td>
</tr>
<tr>
<td>4</td>
<td>female</td>
<td>1 0</td>
</tr>
<tr>
<td>5</td>
<td>male</td>
<td>0 1</td>
</tr>
<tr>
<td>6</td>
<td>male</td>
<td>0 1</td>
</tr>
<tr>
<td>7</td>
<td>male</td>
<td>0 1</td>
</tr>
<tr>
<td>8</td>
<td>female</td>
<td>1 0</td>
</tr>
<tr>
<td>9</td>
<td>female</td>
<td>1 0</td>
</tr>
<tr>
<td>10</td>
<td>male</td>
<td>0 1</td>
</tr>
<tr>
<td>11</td>
<td>male</td>
<td>0 1</td>
</tr>
<tr>
<td>12</td>
<td>male</td>
<td>0 1</td>
</tr>
<tr>
<td>13</td>
<td>female</td>
<td>1 0</td>
</tr>
<tr>
<td>14</td>
<td>female</td>
<td>1 0</td>
</tr>
<tr>
<td>15</td>
<td>female</td>
<td>1 0</td>
</tr>
<tr>
<td>16</td>
<td>male</td>
<td>0 1</td>
</tr>
<tr>
<td>17</td>
<td>female</td>
<td>1 0</td>
</tr>
<tr>
<td>18</td>
<td>female</td>
<td>1 0</td>
</tr>
<tr>
<td>19</td>
<td>female</td>
<td>1 0</td>
</tr>
</tbody>
</table>
Reducing BI Size

- **Binning**
  - [Kou00, SWS04, RSW05]

- **Encoding**
  - [WuBu98, ChIo99]

- **Compressing**
  - Byte-aligned Bitmap Compression [AnZi96]
  - Word-Aligned Hybrid [SWS02, WOS04]
  - Approximate Encoding [ACFT06]
    - may create false positives, additional verification
  - Reordering [JKCKV04, PTF05]
    - computationally very complex
    - reordering heuristics
**Bitmap Compression (1)**

- Byte-aligned Bitmap Compression (BBC)
- Word-Aligned Hybrid (WAH)
- Based on the run-length encoding
  - homogeneous vectors of bits are replaced with a bit value (0 or 1) and the vector length
    - 0000000 1111111111 000 ⇒ 07 110 03

- BBC and WAH
  - a bitmap is divided into words
    - BBC uses 8-bit words
    - WAH uses 31-bit words
Bitmap Compression (2)

- WAH-compressed bitmaps are larger than BBC-compressed ones
- Operations on WAH-compressed bitmaps are faster than on BBC-compressed ones [SWS02, WOS02, WOS04]
- Our further focus is on comparing WAH to our approach
WAH (1)

a) an example bitmap being compressed (5456 bits)

1000000...0000011100001110000011110000000000000000000001111111111111111111111111111011111

31 bits 5394 bits having value "0" 31 bits

b) dividing the bitmap into 31-bits groups

31 bits 31 bits 31 bits

group 1 group 2 group 176

c) merging adjacent homogeneous groups

31 bits 174 * 31 bits 31 bits

group 1 group 2-175 group 176

d) group encoding by means of a 32-bits word

0 100000...0001110000111

31 bits of the first group

bit=0: tail word

run 1

1 0000...0010101110

fill length 174 * 31 bits

bit=0: fill value

bit=1: fill word

run 2

0 001111111...1111011111

31 bits of the last group

bit=0: tail word

Example taken from [StWu07]
1. For low cardinality attributes bitmaps are dense
   - many homogeneous 31-bit words filled with 1
2. For high cardinality attributes bitmaps are sparse
   - many homogeneous 31-bit words filled with 0
3. For medium cardinality attributes
   - the number of homogeneous 31-bit words is lower
   - the compression ratio decreases

✦ A need for bitmap compression technique suitable for medium cardinality attributes
Our Approach: RLH

RLH - the Run-Length Huffman Compression
Based on
- the Huffman encoding
- a modified run-length encoding
Huffman Encoding

Concept
- original symbols from a compressed file are replaced with bit strings
- the more frequently a given symbol appears in the compressed file the shorter bit string for representing the symbol
- encoded symbols and their corresponding bit strings are represented as a Huffman tree
- the Huffman tree is used for both compressing and decompressing
**RLH (1)**

- **Modified run-length encoding**
  - measures and encoded distances between bits of value 1

```
1 0 0 3 0 3 0 0 1 0 0 0
1 0 1 1 1 0 0 0 1 1 0 0 0 1 1 1 0 1 1 1 1
```

**Clients**

<table>
<thead>
<tr>
<th>ID</th>
<th>sex</th>
<th>female</th>
<th>male</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>male</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>female</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>female</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>female</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>male</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>male</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>male</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>female</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>female</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>male</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>male</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>male</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>female</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>female</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>female</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>male</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>female</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>female</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>female</td>
<td>1 0</td>
<td></td>
</tr>
</tbody>
</table>

**Bitmaps encoded this way are input for the Huffman encoding**

- **female**: 100303001000
- **male**: 030020033
Huffman encoding

- step1: computing frequencies of symbols (distances) in encoded bitmaps

<table>
<thead>
<tr>
<th>distance</th>
<th>frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

female: 100303001000
male: 030020033
Huffman encoding

- step 2: building a Huffman tree

- an encoded symbol is represented by a path from the root to a leaf
**RLH (4)**

- **Huffman encoding**
  - **step3: replacing distances with their Huffman codes**

<table>
<thead>
<tr>
<th>distance</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>110</td>
</tr>
<tr>
<td>2</td>
<td>111</td>
</tr>
</tbody>
</table>

**compressed bitmap for sex='female'**

```
110 0 0 10 0 10 0 0 110 0 0
```

The result of modified run-length encoding for bitmap sex='female'
Experimental Evaluation

- Comparing RLH, WAH, and uncompressed bitmaps with respect to
  - bitmap sizes
  - query response times
- Implementation in Java
  - data and bitmap indexes stored on disk in OS files
- Experiments run on
  - PC, AMD Athlon XP 2500+; 768 MB RAM; Windows XP
- Data
  - 2,000,000 indexed rows
  - indexed attribute of type integer
    - cardinality from 2 to 1000
    - randomly distributed values
WAH and RLH: index sizes

- cardinality increases $\Rightarrow$ size of RLH bitmaps increases more slowly than WAH

<table>
<thead>
<tr>
<th>attr.cardinality</th>
<th>$\frac{s_{WAH}}{s_{RLH}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.03</td>
</tr>
<tr>
<td>5</td>
<td>1.42</td>
</tr>
<tr>
<td>10</td>
<td>2.18</td>
</tr>
<tr>
<td>20</td>
<td>3.43</td>
</tr>
<tr>
<td>50</td>
<td>5.18</td>
</tr>
<tr>
<td>100</td>
<td>5.88</td>
</tr>
<tr>
<td>200</td>
<td>6.01</td>
</tr>
<tr>
<td>500</td>
<td>6.68</td>
</tr>
<tr>
<td>1000</td>
<td>7.36</td>
</tr>
</tbody>
</table>
WAH and RLH: response times

Query

\[
\begin{align*}
\text{select ... from ...} \\
\text{where ind_attribute in (v1, v2, ..., v100)}
\end{align*}
\]
Updating RLH Bitmaps

Costly process
- decompressing the whole bitmap
- modifying the bitmap
- compressing the bitmap

Updating a RLH bitmap
- changes frequencies of distances between 1 bits
- creates new distances between 1 bits
- requires building a new Huffman tree

In a DW environment index structures
- are dropped before loading a DW
- are recreated after loading is finished
1. Dividing a bitmap into 1024-bit sections
   - constructing one Huffman tree based on frequencies of distances from all 1024-bit sections

2. Including in the HT all possible distances that may appear in a 1024-bit section
   - non-existing distances have assigned the frequency of 1
RLH1024 Compression (2)

Advantages

- including all the possible distances in the HT eliminates the need of rebuilding the HT after such a bitmap update that results in a new distance
- 1024-bit sections can be read and processed in parallel
- in order to update a bitmap, only an appropriate 1024-bit section has to be read and uncompressed
RLH and RLH1024: index sizes

The graph shows the index sizes for different cardinalities of indexed attributes. The x-axis represents the cardinality of indexed attribute, and the y-axis represents the index size in bytes. The graph includes three lines, each representing a different method: RLH (blue), RLH-1024 (magenta), and WAH (green).
**RLH and RLH1024: response times**

Query

```
select ... from ...
where ind_attribute in (v1, v2, ..., v100)
```
Summary

Alternative bitmap compression technique based on the run-length encoding and on Huffman encoding

- RLH
- RLH1024

Observations

- RLH offers a higher efficiency in accessing data than WAH for attribute cardinality from 5 to 1000
- Bitmaps compressed with RLH are much smaller than corresponding bitmaps compressed with WAH for attribute cardinalities >10
- RLH1024 offers a data access time characteristic similar to RLH, but additionally RLH1024 may better support bitmap updates
Ongoing and Future Work

**Ongoing**
- evaluating the impact of values distribution on WAH and RLH
- evaluating other than 1024-bit partition schemes
- evaluating the efficiency of updating bitmaps in RLH and RLH1024

**Future**
- developing a cost model for RLH
- developing a framework for selecting the most efficient bitmap partition scheme for RLH
- developing a framework for selecting the most efficient bitmap compression technique for a given data characteristic
- experimentally comparing BBC, WAH, RLH, and AE
- integrating RLH into FastBit
References (1)

✦ Binning
  • [RSW05] Rotem D., Stockinger K., Wu K.: Optimizing Candidate Check Costs for Bitmap Indices. CIKM, 2005

✦ Encoding
  • [ChIo99] Chan C.Y., Ioannidis Y.E.: An Efficient Bitmap Encoding Scheme for Selection Queries. SIGMOD, 1999

✦ Compressing
  • BBC
  • WAH
    • [SWS02] Stockinger K., Wu K., Shoshani A.: Strategies for Processing ad hoc Queries on Large Data Sets. DOLAP, 2002
Compressing

- Approximate Compression with Bloom Filters
  - [ACFT06] Apaydin T., Canahuate G., Ferhatosmanoglu H., Tosun, A. S.: Approximate encoding for direct access and query processing over compressed bitmaps. VLDB, 2006

- Reordering
  - [PTF05] Pinar A., Tao T., Ferhatosmanoglu H.: Compressing Bitmap Indices by Data Reorganization. ICDE, 2005

WAH vs. BBC