Space efficient indexes for the big data era

Sidirourgos, L.

Citation for published version (APA):
Sidirourgos, E. (2014). Space efficient indexes for the big data era

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

Download date: 13 Dec 2018
List of Figures

2.1  Example of zonemaps, bitmaps, and imprints indexes. .................. 21
2.2  Column imprint index with compression (23 cachelines). ............... 24
2.3  Prints of column imprint indexes (‘×’ = bit set, ‘.’ = bit unset) and the respective column entropy $\mathcal{E}$. ....................................... 40
2.4  Cumulative distribution of the columns’ entropy $\mathcal{E}$. ............... 41
2.5  Index size and creation time for different types of columns ($x$-axis enumerates the columns, ordered by size). ............................. 43
2.6  Index size overhead % over the size of the columns. .................... 44
2.7  Index size overhead % over column entropy $\mathcal{E}$. .................... 45
2.8  Query time for decreasing selectivity. ..................................... 47
2.9  Cumulative distribution of query times. .................................... 48
2.10 Factor of improvement over scan and zonemap. ........................... 48
2.11 Number of index probes and value comparisons for queries with selectivity between 0.4 and 0.5. .............................................. 50

3.1  In-memory Split Bloom filters for a cold store resident hashfile index. 58
3.2  In-memory Split Bloom filters for a B+tree index. .......................... 59
3.3  In-memory Split Bloom Filters table. ....................................... 60
3.4  Variations of the workload skew used in experiments. .................. 70
3.5  Creation time for different number of keys and bits per key. ........... 71
3.6  False positive ratio for different number of buckets per Bloom filter. 72
3.7  Average cache misses and cpu cycles over 10 million queries. ......... 74
3.8  Distribution of false positive ratio and filter size across the Bloom filters of one index. .................................................. 77
3.9  Evolution over time for a collection of Bloom filters without changing the access pattern. ................................................. 78
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.10</td>
<td>Incremental build every 10,000 lookups with access patterns that shift focus.</td>
<td>80</td>
</tr>
<tr>
<td>3.11</td>
<td>Incremental build of deteriorated Bloom filters because of updates.</td>
<td>81</td>
</tr>
<tr>
<td>4.1</td>
<td>XML Document about persons</td>
<td>91</td>
</tr>
<tr>
<td>4.2</td>
<td>Example of computing $H(&quot;Arthur&quot;)$</td>
<td>96</td>
</tr>
<tr>
<td>4.3</td>
<td>Finite State Machine for double values</td>
<td>99</td>
</tr>
<tr>
<td>4.4</td>
<td>State Combination Table (SCT)</td>
<td>100</td>
</tr>
<tr>
<td>4.5</td>
<td>Indices for Range-Lookup on Typed Values</td>
<td>101</td>
</tr>
<tr>
<td>4.6</td>
<td>String and Double Index Creation Time and Storage Overhead</td>
<td>108</td>
</tr>
<tr>
<td>4.7</td>
<td>Update Time for String and Double Index</td>
<td>110</td>
</tr>
<tr>
<td>4.8</td>
<td>Hash Stability</td>
<td>110</td>
</tr>
<tr>
<td>5.1</td>
<td>A String and Its Posting Lists</td>
<td>116</td>
</tr>
<tr>
<td>5.2</td>
<td>Algorithm Sample</td>
<td>122</td>
</tr>
<tr>
<td>5.3</td>
<td>Signature-based Posting List</td>
<td>124</td>
</tr>
<tr>
<td>5.4</td>
<td>An Overview of QS-Gram</td>
<td>126</td>
</tr>
<tr>
<td>5.5</td>
<td>Global and local data structures</td>
<td>128</td>
</tr>
<tr>
<td>5.6</td>
<td>XMark data with factors 10, 20 to 320</td>
<td>136</td>
</tr>
<tr>
<td>5.7</td>
<td>wiki-meta (44GB) (a-d) Elapsed Time (e-h) Postings Loaded (i-l) Postings Checked</td>
<td>138</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Dataset statistics. ................................................. 38
4.1 Statistical information for the data sets ....................... 105
5.1 Exact Substring Index Comparison ........................... 115
5.2 \(q\)-gram statistics for various datasets ..................... 117
5.3 Index Creation Performance (sec) .............................. 135
5.4 Index Storage Space (GB) ........................................ 136
## List of Algorithms

2.1 Main function to create the column imprints index: `imprints()`  
2.2 Define the number of bins and the ranges of the bins of the histogram: `binning()`  
2.3 Binary search with nested if-statements to locate the bin which a value falls into: `getbin()`  
2.4 Evaluate range queries over the column imprints index: `query()`  
3.1 Split Bloom filters Construction  
3.2 Sizing Split Bloom filters  
3.3 Incremental sizing of a Bloom filter  
4.1 Hash Function $H$  
4.2 Combination Function $C$  
4.3 Index Creation  
4.4 Index Update  
5.1 Selection of Partial Grams $\text{CHOOSEPARTIALGRAMS}(\sigma, q)$  
5.2 Chunk-Wise Processing $\text{CHUNKPROCESSING}(C_i, q)$