Applying Streaming Algorithms to Data at Rest
Mashhood Ishaque, Ligia Nistor and Kevin Backhouse
{mashhood.ishaque, ligia.nistor, kevin.backhouse}@oracle.com
Oracle Corporation, Cambridge, MA, USA

Guided Navigation
- Customers use Oracle Big Data Discovery for finding:
  - Most frequent elements in their datasets
  - Number of elements of a particular kind

Previous Approaches
- Count number of distinct elements using Count aggregator (written in EQL – Endeca Query Language)
  - Uses sorting and grouping: expensive and requires extra materializations of intermediary tables
  - Implemented using in-memory hash table: need 2367 MB to represent 150 million unique elements!
- Use sampling to get approximation of number of distinct values
  - It does not give accurate answer if number of distinct elements is small
  - Very likely that the sample will not contain all different kinds of elements

SpaceSaving and FrequentK
- Internal FrequentK operator based on SpaceSaving [1,2] algorithm
- Finds the top K most frequently occurring elements, ordered by frequency
- Implemented by ApproxCount operator in EQL

Performance and accuracy results
- Each query returns the top 10 elements that satisfy that query
- First query looks for the dominant hashtags in the Twitter dataset
- Second query returns the countries of the tweets
- Third query looks for the most popular Twitter mentions. The results are shown in the table below:

<table>
<thead>
<tr>
<th>#</th>
<th>Value</th>
<th>Count</th>
<th>Value</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YouTube</td>
<td>752808</td>
<td>YouTube</td>
<td>752808</td>
</tr>
<tr>
<td>2</td>
<td>justinbeiber</td>
<td>278185</td>
<td>justinbeiber</td>
<td>278185</td>
</tr>
<tr>
<td>3</td>
<td>wenynchao</td>
<td>200277</td>
<td>wenynchao</td>
<td>200277</td>
</tr>
<tr>
<td>4</td>
<td>NialOfficial</td>
<td>169866</td>
<td>NialOfficial</td>
<td>169866</td>
</tr>
<tr>
<td>5</td>
<td>HarryStyles</td>
<td>113752</td>
<td>HarryStyles</td>
<td>113752</td>
</tr>
<tr>
<td>6</td>
<td>Real_Liam_Payne</td>
<td>102755</td>
<td>Real_Liam_Payne</td>
<td>97534</td>
</tr>
<tr>
<td>7</td>
<td>getlue</td>
<td>93209</td>
<td>getlue</td>
<td>86281</td>
</tr>
<tr>
<td>8</td>
<td>zaynmalik</td>
<td>87139</td>
<td>zaynmalik</td>
<td>78585</td>
</tr>
<tr>
<td>9</td>
<td>JawabUJJUR</td>
<td>73287</td>
<td>JawabUJJUR</td>
<td>67838</td>
</tr>
<tr>
<td>10</td>
<td>Louis_Tomlinson</td>
<td>70140</td>
<td>foursquare</td>
<td>63386</td>
</tr>
</tbody>
</table>

HyperLogLog and ApproxCOUNTDistinct
- We implemented the HyperLogLog algorithm [3,4] as an aggregator called ApproxCOUNTDistinct in EQL, for counting the number of distinct elements
- The aggregator is very inaccurate when there is a small number of distinct values;
  in this case we default to our COUNTDistinct aggregator

Performance and accuracy results
- We only sort a constant number of values (a few thousand)
- We give accurate counts when there is a small number of distinct values
- We return approximate counts when there is a large number of distinct values, but with a provably small error
- We use well studied algorithms that have been shown to perform well in practice
- Streaming algorithms are usually embarrassingly parallel

Benefits

Guided Navigation [5] (multiple properties)

References
1. Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J. Strauss. Space-optimal heavy hitters with strong error bounds. 2010