Information Retrieval

Lecture 9
Outline

• Map Reduce, cont.
• Index compression
• [Amazon Web Services]
Map Reduce

- Map: \((key, value) \rightarrow list(key', value')\)
- Reduce: \((key', list(value')) \rightarrow list (value')\)
Example: counting occurrences of words in large collections of documents

map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
Other Map-Reducible problems

- **Distributed Grep**: The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.

- **Count of URL Access Frequency**: The map function processes logs of web page requests and outputs ⟨URL, 1⟩. The reduce function adds together all values for the same URL and emits a ⟨URL, total count⟩ pair.

- **Reverse Web-Link Graph**: The map function outputs ⟨target, source⟩ pairs for each link to a target URL found in a page named source. The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: ⟨target, list(source)⟩
Compression

• Inverted lists are very large
  – e.g., 25-50% of collection for TREC collections using Indri search engine
  – Much higher if n-grams are indexed

• Compression of indexes saves disk and/or memory space
  – Typically have to decompress lists to use them
  – Best compression techniques have good compression ratios and are easy to decompress

• **Lossless** compression – no information lost
Compression: example

• amendment
  allen-p/_sent_mail/465.:1:34
  stclair-c/sent/993.:5:45,60,76,84,100

• Too verbose!
Compression: example

• amendment
  allen-p/_sent_mail/465.:1:34
  stclair-c/sent/993.:5:45,60,76,84,100

  (that’s 74 characters)

• compare to

• 34
  1342:1:34
  9745:5:45,60,76,84,100

  (that’s 33 characters)

need to keep a mapping between numbers and names
Compression

- **Basic idea:** Common data elements use short codes while uncommon data elements use longer codes
  - Example: coding numbers
    - number sequence: 0, 1, 0, 3, 0, 2, 0
    - possible encoding: 00 01 00 10 00 11 00
    - encode 0 using a single 0: 0 01 0 10 0 11 0
    - only 10 bits, but...
Compression Example

- *Ambiguous* encoding – not clear how to decode
  - another decoding:
    
    $0 01 01 0 0 11 0$

  - which represents:

    $0, 1, 1, 0, 0, 3, 0$

- use unambiguous code:

  - which gives:

    $0 101 0 111 0 110 0$

<table>
<thead>
<tr>
<th>Number</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>111</td>
</tr>
</tbody>
</table>
Delta Encoding

- Word count data is a good candidate for compression
  - many small numbers and few larger numbers
  - encode small numbers with small codes
- Document numbers are less predictable
  - but differences between numbers in an ordered list are smaller and more predictable
- *Delta encoding*:
  - encoding differences between document numbers (*d-gaps*)
Delta Encoding

• Inverted list (without counts)
  1, 5, 9, 18, 23, 24, 30, 44, 45, 48

• Differences between adjacent numbers
  1, 4, 4, 9, 5, 1, 6, 14, 1, 3

• Differences for a high-frequency word are easier to compress, e.g.,
  1, 1, 2, 1, 5, 1, 4, 1, 1, 3, ...

• Differences for a low-frequency word are large, e.g.,
  109, 3766, 453, 1867, 992, ...