Full-Text Indexing For Heritrix

Project Advisor:
Dr. Chris Pollett

Committee Members:
Dr. Mark Stamp
Dr. Jeffrey Smith
Agenda

- Introduction
- Heritrix
- Design and Implementation
- Testing and Results
- Conclusion
Introduction

- The Internet Archive
  - An open source project
  - Stores the archived version of the World Wide Web.
  - Provides domain-based search for the archived web.
- This project takes idea from Internet Archive and extends the search capability on the archive files to provide keyword-based search on the archives for a user.
Heritrix

- An open-source web crawler
- Respects robots.txt exclusion directives and META robots tags
- Uses adaptive crawl method to avoid too much traffic
- It can be obtained from: http://sourceforge.net/projects/archive-crawler/files/
Design and Implementation

- **Two Modules**
  - Indexing Module
    - Iterates through all the archive files and constructs an inverted index for the html files inside these archive files.
    - Calculates BM25F scores partly for relevance of search results based on query terms.
  - Searching Module
    - Provides a Front-end user interface for users to easily search through the index created by the Indexing Module and retrieve desired results.
Indexing Module

- Iterating through Archive files
- Parsing HTML Documents
- Stemmer
- Hashing
- Word Dictionary
- Inverted Index
- BM25F Score
Iterating through archive files

- Heritrix provides access to the classes like ArchiveReader to read through archive files using the API. This class provides iterator of type ArchiveRecord and allows us to iterate through all the records within an archive file.
- ArchiveRecord contains header information about the archive records and the record itself.
Parsing HTML Documents

- HTML documents can be parsed in two ways.
  - **DOM Parsing**
    - Sequential parsing of document
    - Builds tree structure of document in memory
  - **SAX Parsing**
    - Event-based parsing of document
    - Stores data only related to current event or tag being processed
- SAX Parsing is generally more memory efficient compared to DOM Parsing.
- In this project, I used SAX Parser implemented in Java.
Stemming

- Stemming is a process of reducing the derived words to their root forms.
STEMMING helps in grouping related search results together and considering results having words with same root.

In this project, open-source Porter Stemmer developed in Snowball was used.

**Stop-words** have little or no lexical meaning and are used for grammar. They can be safely eliminated from the indexing process to save memory.
Hashing

- MD5 hashing is performed on words and document urls to obtain strings of the same length for word_id and doc_id.
- It uses first 8 characters from the generated MD5 hash code.
Word Dictionary

- It is mapping from word_id generated using MD5 hashing explained previously to the string word.
- All dictionary related data is stored using on-disk Binary Tree data structure implemented in JDBM Project.
Inverted Index

- An inverted index refers to mapping of the word to the set of documents where the word appears.
- It consists of two basic components:
  - Document Dictionary
  - Posting List
**Document Dictionary**

- It contains details for each document in the collection and has mapping from doc_id to the details of the document.

- Followings are the fields stored in Doc Dictionary:

<table>
<thead>
<tr>
<th>Field</th>
<th>Info</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Actual Link to the document</td>
<td>Providing live link in search result</td>
</tr>
<tr>
<td>file</td>
<td>Archive file where it is stored</td>
<td>Locating archive file in order to load cached result from archive file</td>
</tr>
<tr>
<td>offset</td>
<td>Offset at which it is stored in archive file</td>
<td>Locating record referring to the document within archive file</td>
</tr>
<tr>
<td>length</td>
<td>Length of the record in archive file</td>
<td>Determining length of the document</td>
</tr>
<tr>
<td>title</td>
<td>Title of the Document if present</td>
<td>Title of the search result</td>
</tr>
<tr>
<td>desc</td>
<td>Description found in meta tag of the page to generate snippet of the result</td>
<td>Snippet of the search result</td>
</tr>
</tbody>
</table>
- **Posting List**
  - It contains information about occurrence of words within collection of documents and mapping from the word to collection of documents where it appears.
  - Followings are the fields stored in the Posting List:

<table>
<thead>
<tr>
<th>Field</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>word_id</td>
<td>MD5 hashed word to uniquely identify the word</td>
</tr>
<tr>
<td>doc_id</td>
<td>MD5 hashed document URL to uniquely identify the document</td>
</tr>
<tr>
<td>element</td>
<td>HTML element where the word occurred in the document</td>
</tr>
<tr>
<td>count</td>
<td>Frequency of the word in the given document</td>
</tr>
<tr>
<td>length</td>
<td>Length of the HTML element where the word appeared in the document</td>
</tr>
</tbody>
</table>
BM25F Score

- BM25F is used to rank structured documents like an XML or an HTML file.
- The BM25F scoring function gives relevance score of a document for given query terms based on importance of the element in which the terms appear.
- In order to reduce query time calculation for BM25F score, part of the calculation is done at indexing time.
- **Weight**
  - The weight of the term ‘t’ in the document ‘d’ can be calculated as shown in the following equation:

  \[
  \text{weight}(t, d) = \sum_{c \text{ in } d} \left( \frac{\text{occurs}_{t,c}^d \cdot \text{boost}_c}{(1 - b_c) + b_c \cdot \frac{l_c}{\text{av}l_c}} \right)
  \]

<table>
<thead>
<tr>
<th>Term</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>occurs_{t,c}^d</td>
<td>Term frequency of term ‘t’ in document ‘d’ in field ‘c’</td>
</tr>
<tr>
<td>boost_c</td>
<td>Boost factor applied to field ‘c’</td>
</tr>
<tr>
<td>b_c</td>
<td>Constant related to the field length</td>
</tr>
<tr>
<td>l_c</td>
<td>Field length</td>
</tr>
<tr>
<td>avl_c</td>
<td>Average length for the field ‘c’</td>
</tr>
</tbody>
</table>
**Inverse Document Frequency**

Inverse Document Frequency of the term ‘t’ can be derived using following function:

\[ idf(t) = \log \left( \frac{N - df(t) + 0.5}{df(t) + 0.5} \right) \]

<table>
<thead>
<tr>
<th>Term</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Number of documents in the collection</td>
</tr>
<tr>
<td>df(t)</td>
<td>Number of documents where term ‘t’ appears (Document Frequency)</td>
</tr>
</tbody>
</table>
Finally, we use the Weight and Inverse Document Frequency calculated for each query term and use them in below equation to calculate final BM25F score of the document for given query

\[
R(q, d) = \sum_{t \text{ in } q} \frac{\text{weight}(t, d)}{k_1 + \text{weight}(t, d)} \cdot \text{idf}(t)
\]

<table>
<thead>
<tr>
<th>Term</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight(t,d)</td>
<td>Weight of term ‘t’ over all fields for document ‘d’</td>
</tr>
<tr>
<td>(k_1)</td>
<td>Free parameter</td>
</tr>
<tr>
<td>(\text{idf}(t))</td>
<td>Inverse Document Frequency term ‘t’ over collection of documents</td>
</tr>
</tbody>
</table>
Searching Module

- The Query Module
- The Load Cache Module
- The Settings Module
The Query Module

- Main Searching module to type in queries and retrieve results based on that query.
- Retrieves results from index using IndexReader supporting class from Indexing.
The Query Module internals

![Diagram of the Query Module internals]

1. User
2. IndexReader
3. Index
4. 4
5. 5
6. 6
The Load Cache Module

- Takes request from Query Module to load archived version of the selected document.
- Goes to archive files and retrieves the document as it was stored when crawled.
Load Cache Module Working Diagram
The Settings Module

- Provides user with option to change parameters related to the index and archive files.
- The Query Module and The Load Cache Module refers to the settings saved using this module.
## Settings available to users

<table>
<thead>
<tr>
<th>Setting</th>
<th>Option</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobs_path</td>
<td>Path to jobs folder where archive jobs are stored</td>
<td>Jobs/</td>
</tr>
<tr>
<td>job_id</td>
<td>The job id in the jobs_path to refer</td>
<td>01/</td>
</tr>
<tr>
<td>arcs_dir</td>
<td>Path to directory containing archive files relative to jobs_path.</td>
<td>Arcs/</td>
</tr>
<tr>
<td>index_dir</td>
<td>Path to directory containing the index file relative to jobs_path</td>
<td>Index/</td>
</tr>
<tr>
<td>index_name</td>
<td>Name of the index file</td>
<td>MyIndex</td>
</tr>
<tr>
<td>debug_flag</td>
<td>Prints debugging information on console</td>
<td>false</td>
</tr>
</tbody>
</table>
Settings Module Working Diagram
Testing and Results

- Indexing Test:
  - Indexing performance was tested by indexing 38 archive files of size 3.49GB. These archive files were retrieved from a Heritrix crawl ran on April 11, 2011 on sjsu.edu domain.
  - It took around 13887 seconds to create index for 38 archive files of total size 3.49GB. It retrieved 24109 html documents out of those archive files and created inverted index based on them. Size of created index is roughly ~700MB.
Searching Test

- Perform speed test by providing different search queries to retrieve result from index created in Indexing Test.
- Measure number of results, and time required to retrieve the records, build the search result and deliver it to the user.

<table>
<thead>
<tr>
<th>Query</th>
<th>No. of Results</th>
<th>Time Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>google</td>
<td>114</td>
<td>0.232</td>
</tr>
<tr>
<td>mechanical and aerospace engineering</td>
<td>667</td>
<td>0.32</td>
</tr>
<tr>
<td>department of computer science</td>
<td>2782</td>
<td>0.7</td>
</tr>
<tr>
<td>clark hall</td>
<td>238</td>
<td>0.424</td>
</tr>
<tr>
<td>Search</td>
<td>760</td>
<td>0.236</td>
</tr>
<tr>
<td>san jose state university</td>
<td>7561</td>
<td>1.135</td>
</tr>
</tbody>
</table>
Time taken to retrieve result increases as more terms are added in the search query.
Combining results for both graphs shows that the time taken to retrieve results increases as we introduce more terms in search query or there are more results to return.
Conclusion

- Full-Text Indexing for Heritrix, in combination with Heritrix, provides everything one may need to create searchable archives.
- It can be used for archiving different versions of web, and providing keyword-based search for them.
Questions?
Demo!