5/24/2012

Full-Text Indexing For Heritrix

Project Advisor:
Dr. Chris Pollett
Committee Members:
Dr. Mark Stamp
Dr. Jeffrey Smith

Darshan Karia
CS298 – Master's Project Writing

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.

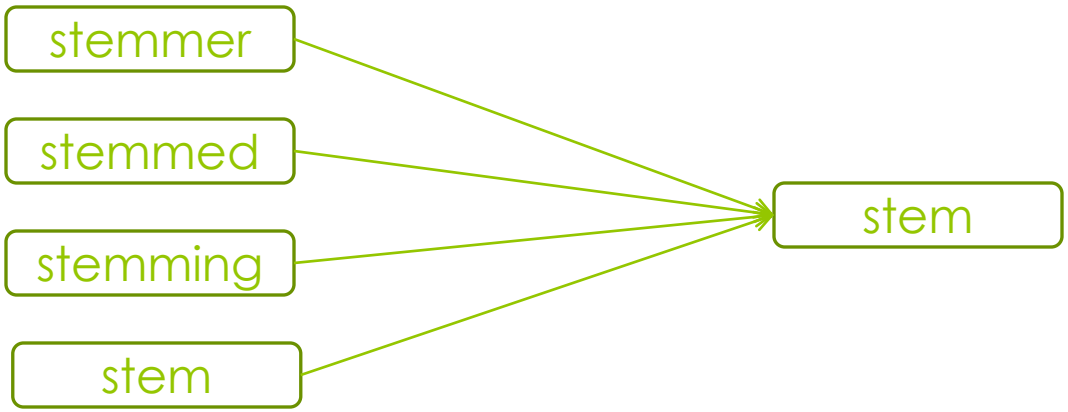
Parsing HTML Documents

- 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.
- E.g. ‘stemmer’, ‘stemmed’, ‘stemming’, ‘stem’, all refers to same root ‘stem’.



- 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:

Field	Info	Used for
url	Actual Link to the document	Providing live link in search result
file	Archive file where it is stored	Locating archive file in order to load cached result from archive file
offset	Offset at which it is stored in archive file	Locating record referring to the document within archive file
length	Length of the record in archive file	Determining length of the document
title	Title of the Document if present	Title of the search result
desc	Description found in meta tag of the page to generate snippet of the result	Snippet of the search result

- **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:

Field	Info
word_id	MD5 hashed word to uniquely identify the word
doc_id	MD5 hashed document URL to uniquely identify the document
element	HTML element where the word occurred in the document
count	Frequency of the word in the given document
length	Length of the HTML element where the word appeared in the document

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:

$$weight(t,d) = \sum_{c\ in\ d} \frac{occurs_{t,c}^{d} \,.\, boost_c}{\left((1-b_c) + b_c \cdot \frac{l_c}{avl_c}\right)}$$

Term	Explanation
$occurs_{t,c}^{d}$	Term frequency of term ‘t’ in document ‘d’ in field ‘c’
$boost_c$	Boost factor applied to field ‘c’
b_c	Constant related to the field length
l_c	Field length
avl_c	Average length for the field ‘c’

- **Inverse Document Frequency**
- Inverse Document Frequency of the term 't' can be derived using following function:

$$idf(t) = \log \frac{N - df(t) + 0.5}{df(t) + 0.5}$$

Term	Explanation
N	Number of documents in the collection
df(t)	Number of documents where term 't' appears (Document Frequency)

- **Final Scoring**
- 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

$$R(q,d) = \sum_{t\ in\ q} \frac{weight(t,d)}{k_1 + weight(t,d)} . idf(t)$$

Term	Explanation
weight(t,d)	Weight of term 't' over all fields for document 'd'
k_1	Free parameter
idf(t)	Inverse Document Frequency term 't' over collection of documents

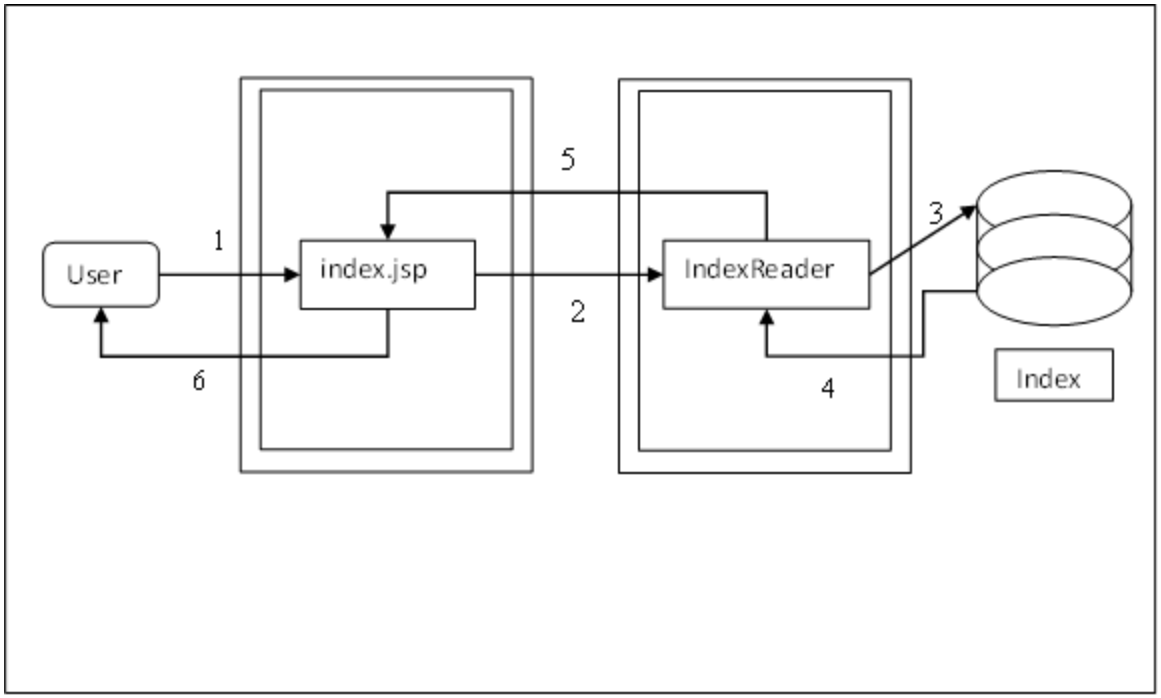
Searching Module

- 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.

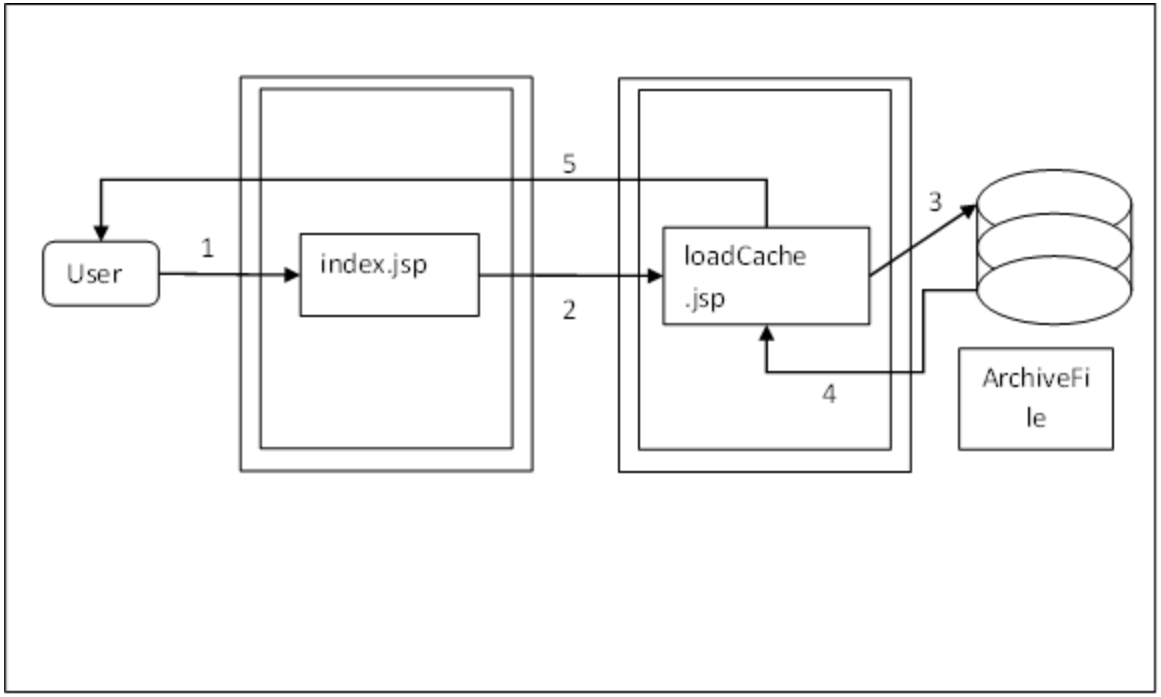
- The Query Module internals



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



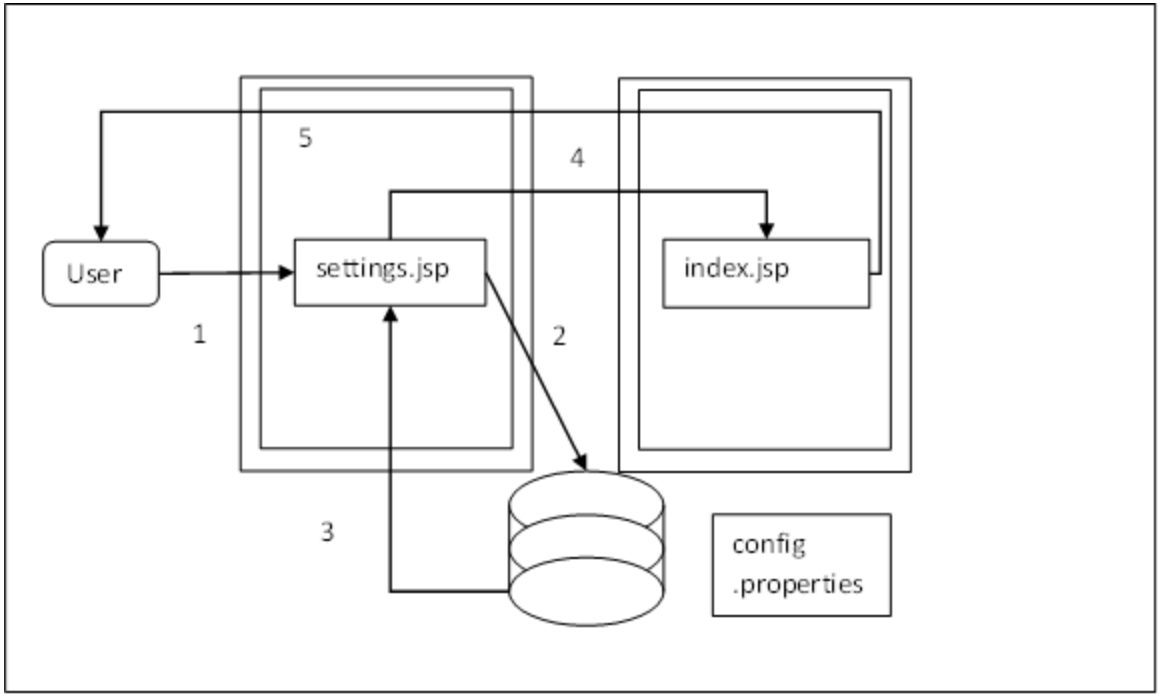
The Settings Module

- 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

Setting	Option	Default Value
jobs_path	Path to jobs folder where archive jobs are stored	Jobs/
job_id	The job id in the jobs_path to refer	01/
arcs_dir	Path to directory containing archive files relative to jobs_path.	Arcs/
index_dir	Path to directory containing the index file relative to jobs_path	Index/
index_name	Name of the index file	MyIndex
debug_flag	Prints debugging information on console	false

- Settings Module Working Diagram



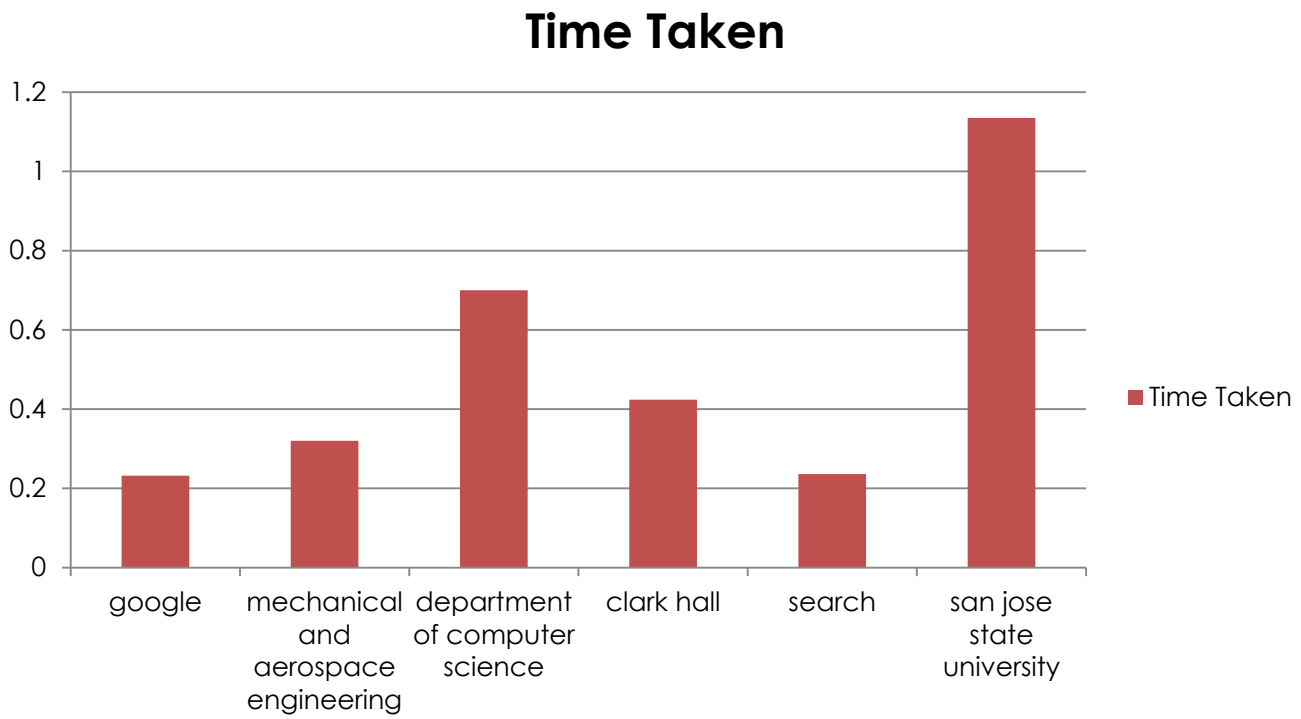
Testing and Results

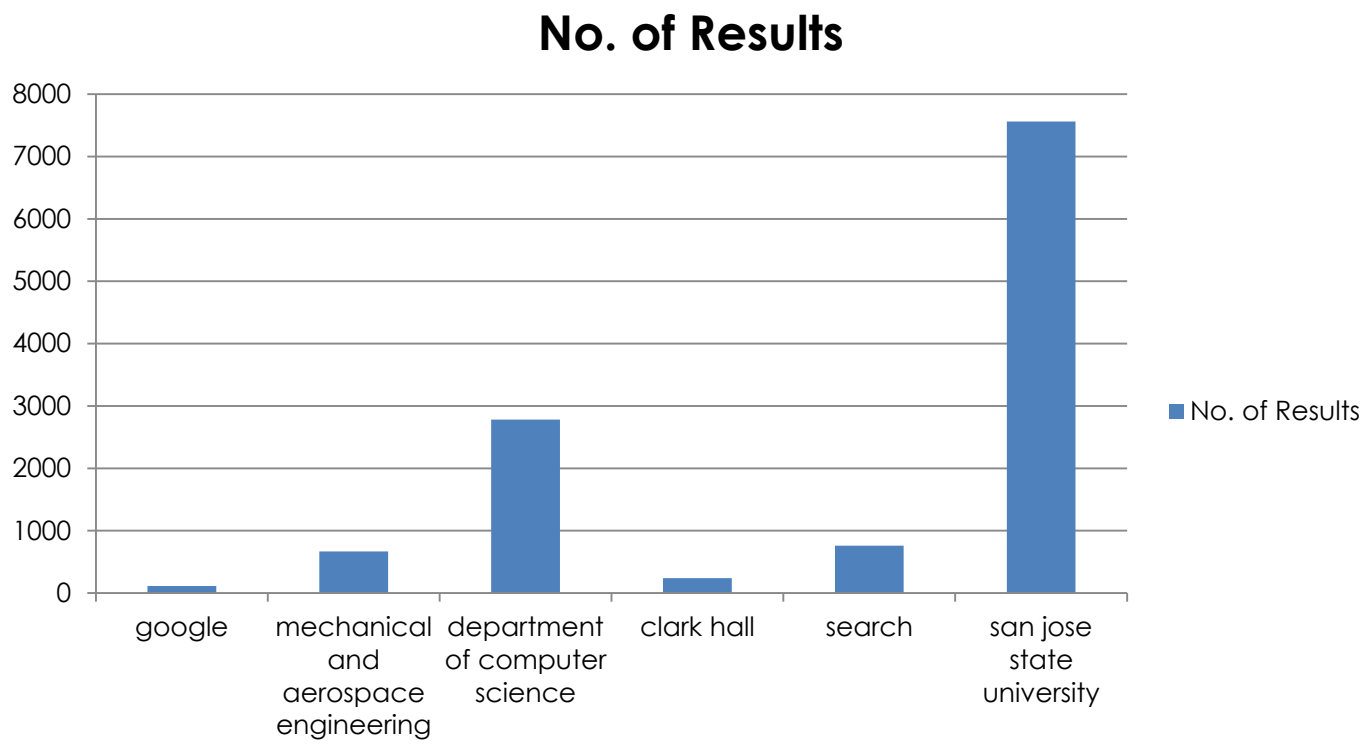
- Indexing Test:
  - Indexing performance was tested by indexing 38 archive files of size 3.49GB. These archive files were retrieved from a Heritrix crawl ran on April 11, 2011 on sjsu.edu domain.
  - It took around 13887 seconds to create index for 38 archive files of total size 3.49GB. It retrieved 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.

Query	No. of Results	Time Taken
google	114	0.232
mechanical and aerospace engineering	667	0.32
department of computer science	2782	0.7
clark hall	238	0.424
Search	760	0.236
san jose state university	7561	1.135

- 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!