ISCTE IUL
Instituto Universitário de Lisboa
Lisbon University Institute

Department of Information Science and Technology

Building a Scalable Index and a Web Search Engine for Music on the Internet using Open Source software

André Parreira Ricardo

Thesis submitted in partial fulfillment of the requirements for the degree of
Master in Computer Science
and Business Management

Advisor: Professor Carlos Serrão, Assistant Professor,
ISCTE-IUL

September, 2010

Acknowledgments

I should say that I feel grateful for doing a thesis linked to music, an art which I love and esteem so much. Therefore, I would like to take a moment to thank all the persons who made my accomplishment possible and hence this is also part of their deed too.

To my family, first for having instigated in me the curiosity to read, to know, to think and go further. And secondly for allowing me to continue my studies, providing the environment and the financial means to make it possible.

To my classmate André Guerreiro, I would like to thank the invaluable brainstorming, the patience and the help through our college years.

To my friend Isabel Silva, who gave me a precious help in the final revision of this document.

Everyone in ADETTI-IUL for the time and the attention they gave me. Especially the people over Caixa Mágica, because I truly value the expertise transmitted, which was useful to my thesis and I am sure will also help me during my professional course.

To my teacher and MSc. advisor, Professor Carlos Serrão, for embracing my will to master in this area and for being always available to help me when I needed some advice. Being a researcher his ideas were always concise and helpful to me.

To all those not named here, but that in some way helped to make this work come true, I would also like to express my gratitude.

Table of Contents

Acknowledgments 2
Table of Contents 3
Terms and Definitions 7
1 Abstract 8
2 Sumário 9
3 Introduction 10
  3.1 Problem Statement....................11
  3.2 Goals....................11
4 Problem-solution Approach 13
  4.1 Analyzing existing music recommendation systems....................13
    4.1.1 Introduction....................13
    4.1.2 Music Recommendation Systems....................13
      4.1.2.1 Amazon MP3....................13
      4.1.2.2 Ella....................14
      4.1.2.3 Grooveshark....................14
      4.1.2.4 iTunes Genius....................14
      4.1.2.5 Last.fm....................15
      4.1.2.6 Spotify....................15
      4.1.2.7 Yahoo! Music....................16
    4.1.3 Overview and Conclusion....................16
  4.2 Solution Approach....................17
5 Bibliographic Research 19
  5.1 Open Source tools for web crawling and indexing State of the Art....................19
    5.1.1 Introduction....................19
    5.1.2 Tools Overview....................20
      5.1.2.1 Aspseek....................23
      5.1.2.2 Bixo....................23
      5.1.2.3 crawler4j....................23
      5.1.2.4 DataparkSearch....................24
      5.1.2.5 Ebot....................24
      5.1.2.6 GNU Wget....................25
      5.1.2.7 GRUB....................25
      5.1.2.8 Heritrix....................26
      5.1.2.9 Hounder....................26
      5.1.2.10 ht://Dig....................27
      5.1.2.11 HTTrack....................27
      5.1.2.12 Hyper Estraier....................28
      5.1.2.13 mnoGoSearch....................28
      5.1.2.14 Nutch....................29
      5.1.2.15 Open Search Server....................30
      5.1.2.16 OpenWebSpider....................30
      5.1.2.17 Pavuk....................31
      5.1.2.18 Sphider....................32
      5.1.2.19 Xapian....................32
      5.1.2.20 YaCy....................33

5.1.3 Overview....33
5.1.4 Conclusion....34
6 Methodology 36
7 Proposal 37
8 Validation & Assessment 39
8.1 Gathering Seed URLs....39
8.2 Configuring Nutch....40
8.3 Crawl and Indexing....40
8.4 Parsing and Indexing MP3....41
8.4.1 Extending the index plugin 42
8.4.2 Building the index plugin....42
8.5 Speeding up the fetch....43
8.6 Browsing the index....43
8.7 Searching with Nutch....44
8.8 Boosting fields in the search engine ranking system....45
8.9 Integration with Solr....47
8.9.1 Configuration....48
8.9.2 Queries....48
8.10 Project Management....49
9 Conclusions 50
10 Bibliography 52
11 Annex 56
11.1 Python web scrapping file clongclongmoo.py....56
11.2 Nutch configuration file nutch-site.xml....56
11.3 Nutch plugin index-mp3....59
11.3.1 Plugin structure....59
11.3.2 Indexing filter Mp3IndexingFilter.java....59
11.3.3 build.xml....62
11.3.4 plugin.xml....62

Index of Tables

Table 1: Sample of sites that provide music freely and the number of items for each site....10
Table 2: Open Source tools for web crawling and indexing: short description and noticeable users....20
Table 3: Open Source tools for web crawling and indexing: programming language, index type and database....21
Table 4: Open Source tools for web crawling and indexing: front-end, plugin, MP3 and Flash support....22

Images Index

Figure 8.1: The crawling and indexing process...41
Figure 8.2: Screenshot from Luke showing a MP3 document...43
Figure 8.3: Search in Nutch by "carol"...44
Figure 8.4: Explanation on the score attributed to the first result in Figure 8.3............45
Figure 8.5: Search in Nutch by "carol", with different ranking order from Figure 8.3. 46
Figure 8.6: Explanation on the score attributed to the second result for the query "carol"...47
Figure 8.7: Query by album name using Solr with response in XML............................49

Terms and Definitions

Ajax	Asynchronous JavaScript and XML
API	Application Programming Interface
BPM	Beats Per Minute
CGI	Common Gateway Interface
CLI	Command Line Interface
Crawl	parse through the HTML structure of a page
Cross-platform	Software designed to work on various operating systems
Flash	Adobe Flash
GUI	Graphical User Interface
Hadoop	Software framework for distributed computing and data storage
HDFS	Hadoop Distributed File System
HTML	HyperText Markup Language
ID3	Metadata container for the MP3 audio file format
MP3	MPEG-2 Audio Layer III
OS	Operative System
P2P	Peer-to-Peer
SQL	Structured Query Language
SWF	ShockWave Flash
URL	Uniform Resource Locator
XML	Extensible Markup Language

1 Abstract

The Internet has made possible the access to thousands of freely available music tracks with Creative Commons or Public Domain licenses. Actually, this number keeps growing every year.

In practical terms, it is very difficult to browse this music collection, because it is wide and disperse in hundreds of websites.

To address the music recommendation issue, a case study on existing systems was made, to put the problem in context in order to identify necessary building blocks.

This thesis is mainly focused on the problem of indexing this large collection of music. The reason to focus on this problem, is that there is no database or index holding information about this music material, thus making this research on the subject extremely difficult.

In order to figure out what software could help solve this problem, the state of the art in “Open Source tools for web crawling and indexing” was assessed.

Based on the conclusions from the state of the art, a prototype was developed and implemented using the most appropriate software framework. The created solution proved it was capable of crawling the web pages, while parsing and indexing MP3 files. The produced index is available through a web search engine interface also producing results in XML format.

The results obtained lead to the conclusion that it is attainable to build a scalable index and web search engine for music in the Internet using Open Source software. This is supported by the proof of concept achieved with the working prototype.

Keywords: Content Analysis and Indexing, Information Storage and Retrieval, Information Filtering, Retrieval Process, Selection Process, Open Source, Creative Commons, Music, MP3

2 Sumário

A Internet tornou possível o acesso a milhares de faixas musicais disponíveis gratuitamente segundo uma licença Creative Commons ou de Domínio Público. Na realidade, este número continua a aumentar em cada ano.

Em termos práticos, é muito difícil navegar nesta colecção de música, pois a mesma é vasta e encontra-se dispersa em milhares de sites na Web.

Para abordar o assunto da recomendação de música, um caso de estudo sobre sistemas de recomendação de música existentes foi elaborado, para contextualizar o problema e identificar os grandes blocos que os constituem.

Esta tese foca-se na problemática da indexação de uma grande colecção de música, pela razão de que, não existe uma base de dados ou índice que contenha informação sobre este repositório musical, tornando muito difícil o estudo nesta matéria.

De forma a compreender que software poderia ajudar a resolver o problema, foi avaliado o estado da arte em ferramentas de rastreio de conteúdos web e indexação de código aberto.

Com base nas conclusões do estado da arte, o protótipo foi desenvolvido e implementado, utilizando o software mais apropriado para a tarefa. A solução criada provou que era possível percorrer as páginas Web, enquanto se analisavam e indexavam MP3. O índice produzido encontra-se disponível através de um motor de busca online e também com resultados no formato XML.

Os resultados obtidos levam a concluir que é possível, construir um índice escalável e motor de busca na web para música na Internet utilizando software Open Source. Estes resultados são fundamentados pela prova de conceito obtida com o protótipo funcional.

Palavras-Chave: Análise e Indexação de Conteúdos, Armazenamento e Recuperação de Informação, Processo de Disponibilização, Filtração de Informação, Open Source, Creative Commons, Música, MP3

3 Introduction

Currently there are thousands of music tracks and music albums available on the Internet for free and it is predictable, in the near future, there will be millions (Table 1) (Metrics, 2010). All the music tracks and albums listed (Table 1), are under a Creative Commons (CC) License (Creative, 2010) which allows free sharing. With this CC license they can be downloaded and shared free of charge and legally. It is often referred as “free music” or Netaudio (Netlabel, 2010).

Since today it is easy and common to share audio files, the Creative Commons have earned a growing relevance (Metrics, 2010), because they are more focused and built for this new reality of sharing than the traditional copyright license, which is much more restrictive and limits more the user who seeks to share and distribute freely its contents and also wishes others to do the same (Lessig 2004). Besides the copyright license not allowing sharing, it also does not permit the use for non-commercial purposes. On the other hand, Creative Commons licenses marked as “non-commercial” cannot be used in a commercial context but they may be used in a home made movie soundtrack, for instance. This makes them quite attractive with the growth in personal digital home video. This is just one example out of many that can be provided through the use of Creative Commons licenses.

Site	Items at 2009-06-08	Items at 2010-01-04
Archive.org[1]	348000	460796 audio items
Jamendo.com[2]	20542	28740 published albums
SoundClick.com[3]	510,370 CC tracks	537,892 CC tracks
Magnatune.com[4]	8872 tracks	9624 tracks
testtube.monocromatica.com[5]	1037 tracks (172 albums)	196 albums
lastima.net[6]	35 tracks	35 tracks

Table 1: Sample of sites that provide music freely and the number of items for each site

1 http://www.archive.org/details/audio
2 http://www.jamendo.com/en/
3 http://www.soundclick.com/business/license_list.cfm Accessed: 2010-01-04, in the box “License type” choose “Creative Commons (free)”
4 http://magnatune.com/info/stats/ Accessed: 2010-01-04
5 http://testtube.monocromatica.com/index.htm Tracks counted in 2009-06-08
6 http://www.lastima.net/ Accessed: 2010-01-04

Another example that could be provided is the Flickr Web site[7] which hosts more than 100 million[8] photos licensed under Creative Commons that were uploaded to the site[9] by its users and that are this way employed by blog authors looking for pictures to use in their work.

3.1 Problem Statement

At the present moment, if one wants to conduct a research based on the music freely available on the internet there is not a database or index holding the records for this content. If in Flickr users can find a good photo by looking at a vast database of indexed content, in the free music World such "library" simply does not exist.

Without this "library" or index, the enormous range of uses to Creative Commons music is being wasted. Users are not finding songs for their leisure or to use in their projects, and artists are kept unknown without gaining any visibility in their work, despite the fact that they are publishing under a license that facilitates content sharing.

There is the need to deal with this issue but it is also important to notice, because this collection is already so big (Table 1), that the problem must be addressed in early stages with scale in mind, so that future work may not be limited by the collection growing size.

The identified problem can be put in the form of the following question "How to index the existing music collection on the Internet in a scalable way?"

3.2 Goals

To handle the issue of the inexistent index or database for the music freely available on the Internet, an exploratory study will be conducted to describe systems that crawl, index, manage and preserve content and metadata in a large scale. This thesis will carry out some research on how to build this index/database for the sparse free music collection on the internet.

However the research on these systems to store and maintain the content, was limited to open source software for the following reasons:

- the goal is to create a solution for the long term that will be the basis for future work. With open-source stands the will to be independent from a vendor solution, limiting the vendor lock-in problem and limiting the risk of a future solution being

7 http://www.flickr.com/ Accessed in: 2010-01-08
8 http://creativecommons.org/weblog/entry/13588 Accessed in: 2010-01-08
9 http://www.flickr.com/creativecommons Accessed in: 2010-01-08

compromised if for any reason the vendor leaves the market;

- in general, there is less information and details one can find about commercial solutions inner workings, making it more difficult to persue a serious research about them;
- necessity to meddle and make changes to the software in order to tune it to the project's reality and context, only possible with open source;
- this is a work developed in an academic context without financial support to acquire commercial software.

In a second phase an exploratory study will be conducted about existing open-source tools for web crawling and indexing. This study will lead to the analysis of the a state of the art.

At the end, the created solution will be validated and assessed based on the implemented prototype.

This thesis is written in English because the world lives in an era of globalization. The goal is to make this document available to a larger number of people in a way that can open new perspectives on this work.

4 Problem-solution Approach

The problem-solution approach is divided into two different parts. The first part, “Analyzing existing music recommendation systems”, presents an exploratory study on systems that have a large collection of music MP3 files available on the Web. In the second part, “Solution Approach”, the existing problem will be put in context and divided to find a solution to the music recommendation problem.

4.1 Analyzing existing music recommendation systems

4.1.1 Introduction

In this analysis the aim is look at to existing recommendation systems handling large collections of music and MP3 or similar indexed files.

This is a descriptive study to understand how music recommendation systems work. The aim is to help put into context the work being done in section 5.1, once the objective is to put in context the music recommendation problem.

It is important to note that only services where the user can select precisely which music is willing to listen, were considered. Radios and systems limited to recommending similar songs, while not playing the actual songs sought by the user are out of the scope, as they do not provide an index searchable by the user.

4.1.2 Music Recommendation Systems

4.1.2.1 Amazon MP3

Amazon, the popular online retail shop, uses its famous collaborative filtering item-to-item algorithm applied to music recommendation (Aucouturier and Pachet 2002).

The item-to-item filtering focus the recommendation on the item rather than the user. The basic principle of this algorithm is based on the acception that “users who bought x also bought y”. In this case, it is recommended to the user similar items in terms of what is the users' buying/listening experience. For more information on Amazon.com item-to-item Collaborative Filtering see (Linden, Smith, and York 2003)

4.1.2.2 Ella

Ella is the recommendation system from BMAT, a commercial spin-off of the Music Technology Group, the world's largest research lab in music and audio that combines together a set of algorithms and it is the result of the work being done in the last 8 years that enable a computer to describe and understand music.

The Ella system uses an hybrid approach to recommendation using algorithms to automatically detect tempo, BPM, chord progression, key, tonality, instrumentation and mood that were perfected and combined with context information (country, label, genre, recording date, reviews, concerts, and others) and even with moods, just to make a search engine solution that is smart and human-like (Bertin-mahieux et al. 2008).

Relevant work from the authors that match Ella's description can be seen in (Oscar Celma et al. 2009).

4.1.2.3 Grooveshark

Grooveshark is an online search music engine and music streaming service. Currently Grooveshark has one of the largest full length streaming databases with 5 million songs available for mobile devices (Grooveshark 2010). Users can contribute by uploading music to the site.

The site technology stack consists mostly on open source software, running Linux, Apache, Lighttpd, MySQL, Sphinx full text indexes and more (Paroline 2010). The service also uses, Hive, a data warehouse infrastructure built on top of Hadoop. Hadoop is used for " user analytics, dataset cleaning, and machine learning R&D." (Hive 2010)

Recently the site added a radio featuring music recommendation with possible feedback from users on the results, clicking on a happy smile or a sad smile icon.

4.1.2.4 iTunes Genius

The iTunes Genius is Apple's Computer music recommendation system bundled with iTunes music player/manager. For iTunes the database problem was solved recurring to Gracenote so "when iTunes users are adding music to their collection or looking for recommendations the answers come straight from Gracenote." (Gracenote 2010)

According to (Luke Barrington; Reid Oda; Gert Lanckriet 2009) iTunes Genius appears to use collaborative filtering to compare the seed song's metadata to the iTunes' massive database of music sales (over 50 million customers who have purchased over 5 billion songs), as well as the play history and song rating data collected from iTunes users.

4.1.2.5 Last.fm

Last.fm is one of the most popular music recommendation sites commercially available. Born from the Audioscrobbler recommendation system, Last.fm uses the information gathered from user listening habits (O. Celma 2006). "Scrobbling" is the process in which the user's music player uploads information to the Last.fm site, referring to what song is currently being played and how it is called/referred by others.

The database around Last.fm is created by "scroobles" from users. When a track is submitted, if the artist does not exist in the database, it is automatically inserted. A page for the artist is created and the users can fill the missing information by adding a description or uploading photos.

To support this immense database and computation needs Last.fm relies partly on Hadoop (Dittus 2008).

Using collaborative filtering techniques, Last.fm provides recommendations of similar artists based on the user's previous listened artists. Using tag information applied to each artist, Last.fm can also recommend artists with similar tags. A radio station based on the user listening habits is also automatically created to provide further recommendations.

A broad set of services, companies and other recommender systems use the Last.fm API to get information regarding the user and artist data, especially artists' music tags. There is also a Web radio platform with over 7 million tracks available (Richard 2009).

4.1.2.6 Spotify

Spotify is a desktop application where peer-to-peer music is used for a streaming service with unlimited streaming and a free with ads subscription or a paid ad-free subscription.

The Spotify recommendation system was built, specifically for this Swedish company, so that "users can listen to a vast collection of several million tracks, streaming over the Internet." (Bernhardsson 2009)

According to work done at Spotify, it is important to define metrics as simple as possible when evaluating recommender system performance and where human feedback is not possible. Instead of trying to optimize specific metrics and using only one algorithm, focus should be put on deriving fundamentally different algorithms (Bernhardsson 2009).

There is further academic research supporting the idea of not spending too much effort optimizing specific metrics (McNee, Riedl, and Konstan 2006).

To support large scale, low latency, P2P music-on-demand streaming, Spotify relies Hadoop for log analysis (Kreitz and Niemela 2010).

4.1.2.7 Yahoo! Music

Yahoo! Music uses a similar approach to Amazon.com collaborative filtering techniques.

Yahoo Research Labs tackled the problem of recommending music to a group of friends using an open source collaborative filtering engine. (Decoste et al. 2005)

To support Yahoo! Music, Hadoop clusters similar to the ones used by Yahoo should be in use (Yahoo 2008) (Yahoo 2010) (Shvachko et al. 2010).

4.1.3 Overview and Conclusion

With this work it is possible to understand that there are three different types of recommendation techniques: using only audio content analysis, applying collaborative filtering to the users usage and feedback, or combining both in an hybrid approach.

In terms of the software used, it is important to notice the common usage of the Hadoop, a software framework for scalable and distributed computing, in four of the seven analyzed cases. Another aspect to retain is the possibility to build the support framework for these large scale systems recurring to open source software.

Placing the music recommendation problem in context, allows the topic to be divided, at least, into the following topics:

- Indexing
- Content Analysis Algorithms
- Collaborative Filtering

Addressing and partitioning the problem this way, will be helpful in following section (4.2)to encounter a viable solution for the problem in question.

4.2 Solution Approach

Recommendation systems can be seen, in terms of a big picture, as a process that flows, from indexing to processing through content analysis algorithms followed by collaborative filtering. The problem-solution approach was: started from this bigger issue, music recommendation, divide it into several key components and then address the first unsolved piece that limited future work.

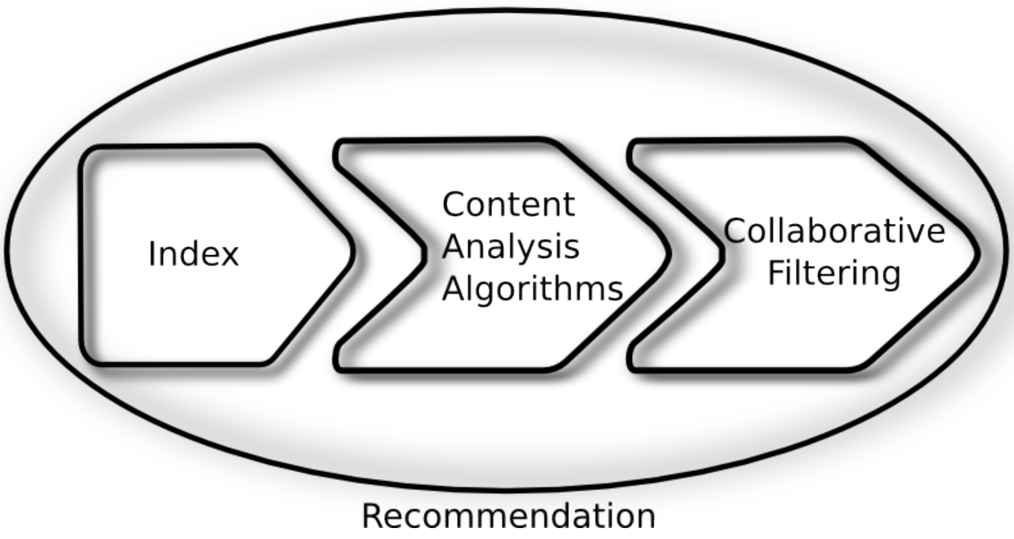


Figure 1: High level schematics on a possible recommendation system

In the example portrayed in Figure 1, music or other data content comes from an index or database and then it is processed by music extraction and analysis algorithms or by collaborative filtering algorithms. Then the content can be recommended, using only one algorithmic approach or by combining both approaches in a third step, calling it a hybrid approach.

By accessing the recommendation problem it was visible that the first element in the chain is currently unsolved for the music collection on the web. This problem is impairing future research from being done because there is no database or index to build upon, the collection is fragmented in the web and unsuitable to do research.

The scope of this work is focused on indexing and developing a music search engine.This a crucial step to allow all the sparse free music available in multiple Web sites to be collected and indexed. This index and data is of the first and utmost importance in any music recommender system. Without the index there is no content available for the recommender system or content analysis algorithms to work on. The solution for this problem is to build a solid indexing base that may be used in the future.

But how to tackle this first problem of gathering from the Internet all the available information relevant to the work addressed by this thesis. And how to index it in a way that is useful for the community but also for researchers working in these fields?

The tool for crawling through all the web pages and following the links from it is called a Web crawler. The process performed by the Web crawler is referred as web crawling or simply “crawl”. It is important for this work that the crawler provides relevant data-structures after crawling. This means that indexing data in a searchable way is a considerable advantage when choosing the appropriate tool for this job.

There are more open source crawling tools available than the ones listed in the following sections, but all the tools without any development or updates in the recent years (last 8 years) or with a really specific scope (for example to parse only some defined type of content) were not considered. The Web is constantly evolving with new changes to the HTML structure and tags (for example the new tags added with HTML5) which outdated tools cannot cope with and therefore were discarded from this study. Having the ability to work with recent web crawlers still being developed and maintained, the preference was given to these tools capable of crawling the ever changing Web.

5 Bibliographic Research

5.1 Open Source tools for web crawling and indexing State of the Art

5.1.1 Introduction

The objective of this section is to identify and study the tools that can be used to create a similar index to the ones used by existing commercial music recommendation systems, introduced at section 4.1, but with the purpose of indexing all freely available music in the Internet.

In the following section it will be depicted a summary of all the existing tools that can be used for this purpose (Table 2) considering all the available projects.

Also in this section, data is presented about all the most important characteristics like programming language, index type, database integration, front-end, plugin structure, MP3 and Flash parsing support are listed among tools.

Concluding the analysis, for each tool the most relevant key advantages and minuses are stated, followed by an overview on how adequate the tool is to solve the problem addressed by this work.

5.1.2 Tools Overview

In this section it will be presented a summary of the different traits of each tool, being primarily displayed in a set of tables to ease the comparison process. The first table (Table 2) introduces all the tools being analyzed with a short description and stating the most notable users operating with each piece of software.

Name	**Description**	**Notable Users**
Aspseek	ASPseek consists of an indexing robot, a search daemon, and a CGI search frontend	
Bixo	web mining toolkit that runs as a series of Cascading pipes on top of Hadoop	Bebo, EMI Music, Share This, Bixo Labs
crawler4j	Java Crawler which provides a programmable interface for crawling	
DataparkSearch	web crawler and search engine	News Lookup
Ebot	web crawler written on top of Erlang	
GNU Wget	non-interactive command line tool to retrieve files from the most widely-used internet protocols	
GRUB	web crawler with distributed crawling	Wikia
Heritrix	extensible, web-scale, archival-quality web crawler project	Internet Archive, Arquivo da Web Portuguesa
Hounder	crawler, indexer and web search engine	
ht://Dig	Search Engine and Web Crawler	
HTTrack	website mirror tool	
Hyper Estraier	full-text search engine system	GNU Project
mnoGoSearch	web search engine	MySQL
Nutch	web search, crawler, link-graph database, parsers, plugin system	Creative Commons, Wikia Search
Open Search Server	search engine with support for business clients	
OpenWebSpider	web spider for the .NET platform	
Pavuk	web crawler	
Sphider	PHP search engine	
Xapian	Search engine, uses ht://Dig for crawling	gmane
YaCy	free distributed search engine, built on principles of peer-to-peer networks	Sciencenet

Table 2: Open Source tools for web crawling and indexing: short description and noticeable users

Considering all the software pieces in analysis, (Table 3) states the programming language used for their development (language) then the platforms in which they run, if there is some type of indexing done by web crawling tools (index) and finally possible connections to databases are also considered (database).

Name	**Language**	**Platform**	**Index**	**Database**
Aspseek	C++	Linux	Relational DB	SQL, binary
Bixo	Java	Cross-platform	N/A	Possible integration
crawler4j	Java	Cross-platform	N/A	
DataparkSearch	C	Cross-platform	SQL	MySQL, PostgreSQL[10]
Ebot	Erlang, NoSQL	Linux	NoSQL	CouchDB
GNU Wget	C	Linux	File mirror	
GRUB	C#	Cross-platform	Relational DB	MySQL
Heritrix	Java	Unix	Arc files	
Hounder	Java	Cross-platform	Lucene	
ht://Dig	C++	Unix	disk files	
HTTrack	C/C++	Cross-platform	Mirror files	
Hyper Estraier	C/C++	Cross-platform	QDBM[11]	
mnoGoSearch	C	Windows	Relational DB	MySQL, PostgreSQL, SQLite
Nutch	Java	Cross-platform	Lucene	
Open Search Server	C/C++, Java PHP	Cross-platform	Lucene	
OpenWebSpider	C#, PHP	Cross-platform	Relational DB	MySQL
Pavuk	C	Unix	Mirror files	
Sphider	PHP	Cross-platform	Relational DB	MySQL
Xapian	C++	Cross-platform	Omega	
YaCy[12]	Java	Cross-platform	NoSQL	

Table 3: Open Source tools for web crawling and indexing: programming language, index type and database

10 http://www.dataparksearch.org/wiki/index.php/Indexing_in_general Accessed: 2010-09-20
11 QDBM (Quick Database Manager) http://fallabs.com/qdbm/index.html Accessed: 2010-09-20
12 http://yacy.net/Technology.html Accessed: 2010-09-20

The following table (Table 4) summarizes for each tool:

- front-end capabilities;
- support for plugins and;
- MP3 or Adobe Flash parsing support.

Flash support is important to address, due to the architecture of some netlabel sites which use exclusively this technology without any HTML link structure to navigate or with links to music files directly inside Flash audio players.

The goal is to understand the extensibility, flexibility and maintainability of each solution.

Name	**Front-end**	**Plugin**	**MP3**	**Flash**
Aspseek	CGI	external converter programs[13]	N/A	N/A
Bixo	Cascading pipes			
crawler4j	API			
DataparkSearch	N/A	External Parsers	built-in	Via external parsers
Ebot	Web Services	Extensible	N/A	N/A
GNU Wget	CLI			
GRUB	PHP			
Heritrix	CLI, JSP			
Hounder	JSP	Uses Nutch Plugins		
ht://Dig	CGI	External Parsers		Via external parsers
HTTrack	GUI			Follow links
Hyper Estraier	CGI	API		
mnoGoSearch	CGI, PHP, Perl		built-in	
Nutch	CLI, JSP	Plugins system	deprecated	
Open Search Server	Web based			
OpenWebSpider	CLI, Web based		UltraID3Lib	
Pavuk	CLI			
Sphider	PHP	External Parsers		
Xapian	CGI, XML	Uses Omega	N/A	N/A
YaCy	Web based			

Table 4: Open Source tools for web crawling and indexing: front-end, plugin, MP3 and Flash support

13 http://www.aspseek.org/man/index.1.php Accessed: 2010-09-20

5.1.2.1 Aspseek

URL: http://www.aspseek.org/

1.1 Overview

There is little information about Aspseek, the tool is outdated and in this scenario it is not a reliable tool to work with.

1.2 Advantages

- External parsers supported

1.3 Minuses

- Outdated, last update in 2002;
- Not scalable for whole web crawl, it is based on a relational database.

5.1.2.2 Bixo

URL: http://openbixo.org

2.1 Overview

Bixo is a tool that might be very interesting to projects looking for a web mining framework that can be integrated with existing information systems. For example to inject data into a data-warehouse system.

Based on the Cascading API that runs on an Hadoop Cluster, Bixo is suitable to crawl large collections. In a project that has the need to handle large collections and to input data into existing systems, Bixo is a tool to have a close look at.

2.2 Advantages

- Oriented to data mining;
- Proof of concept with the Public Terabyte Dataset (DATAMINING, 2009), (BIXO, 2010)

2.3 Drawbacks

- Little built-in support to create an index;

5.1.2.3 crawler4j

URL: http://code.google.com/p/crawler4j/

3.1 Overview

Crawler4j is a piece of source code to incorporate in a project but there are more suitable tools to index content.

3.2 Advantages

- Easy to integrate in Java projects that need a crawling component;

3.3 Drawbacks

- No support for robots.txt neither for pages without UTF-8 encoding;
- It is necessary to create all the complementary framework for indexing.

5.1.2.4 DataparkSearch

URL: http://www.dataparksearch.org/

4.1 Overview

DataparkSearch is a tool that benefits from the stated MP3 and Flash parser but unfortunately, due to lack of development, it is still using outdated technology like CGI and does not have a modular architecture making it difficult to extent. The index is not in a format that could be used by other frameworks.

4.2 Advantages

- Support for MP3 and Flash parser.

4.3 Drawbacks

- Using CGI
- No development in recent times.

5.1.2.5 Ebot

URL: http://www.redaelli.org/matteo-blog/projects/ebot/

5.1 Overview

There is no proof of concept that Ebot would scale well to index the desired collection. Because Erlang and CouchDB were used to solve the crawl and search problem, people keen on these languages might find this tool attractive.

5.2 Advantages

- Ebot is distributed and scalable (Ebot 2010).

5.3 Drawbacks

- Only one developer active in the project;
- There is not a proven working system deployed.

5.1.2.6 GNU Wget

URL: http://www.gnu.org/software/wget/

6.1 Overview

Wget is a really useful command line tool to download a simple HTML website, but it does not offer indexing support. It is limited to the mirroring and downloading process.

6.2 Advantages

- With simple commands it is easy to mirror an entire website or to explore the whole site structure;

6.3 Drawbacks

- There is the need to create all the indexing infrastructure;
- It is build for pages mainly working with HTML with no Flash or Ajax.

5.1.2.7 GRUB

http://grub.org/

7.1 Overview

Grub distributed solution requires a proof of concept that is suitable for a large scale index. It also requires to prove that distributed crawling is a better solution than centralized crawling.

7.2 Advantages

- Tries a new approach to searching by distributing the crawling process.

7.3 Drawbacks

- Documentation incomplete;

- Was banned from Wikipedia for bad crawling behavior;
- According to the Nutch FAQ distributed crawling may not be a good deal, while it saves bandwidth in the long run this saving is not significant. Because it requires more bandwidth to upload query results pages, "making the crawler use less bandwidth does not reduce overall bandwidth requirements. The dominant expense of operating a large search engine is not crawling, but searching."[14]
- Lack of news since 2009. The project development looks halted.

5.1.2.8 Heritrix

URL: http://crawler.archive.org/

8.1 Overview

Heritrix is the piece of software used and written by The Internet Archive to make copies of the internet.

The disadvantage for Heritrix is the lack of indexing capabilities, the content is stored in ARC files (13. Internet 2010).

It is a really good solution to archiving websites and make copies for future reference.

8.2 Advantages

- Software use case proven by Internet Archive;
- Really adjusted to make copies of websites.

8.3 Drawbacks

- Necessity to process Arc files;
- The architecture is more monolithic and not designed to add parsers and extensibility.

5.1.2.9 Hounder

URL: http://hounder.org/

9.1 Overview

Although Hounder is an open source product it suffers from the same problems

14 http://wiki.apache.org/nutch/FAQ#Will_Nutch_use_a_distributed_crawler.2C_like_Grub.3F Accessed: 2010-09-22

pointed to commercial solutions. Hounder is a product from Flaptor and since 2008 there have been no development or updates. Similar solutions like Nutch have evolved while Hounder development halted.

9.2 Advantages

- Hounder claims to be a complete solution for the indexing and searching.[15]

9.3 Drawbacks

- Dependance on Flaptor for future development;
- Development halted circa 2008.

5.1.2.10 ht://Dig

URL: http://www.htdig.org/

10.1 Overview

ht://Dig is a searching system towards generating search for a website. Like a website already built in HTML that wants to add searching functionality.

Until 2004, date of the last release, it was one of the most popular web crawlers and search engine, enjoying a large user base with notable sites such as the GNU Project and Mozilla Foundation but with no updates over the time, slowly lost most of the user base to newer solutions.

10.2 Advantages

- htt://Dig was until 2004 one of the most popular web crawlers and search engine, enjoying a large user base with notable sites such as the GNU Project, Mozilla Foundation

10.3 Drawbacks

- Development stopped circa 2004.

5.1.2.11 HTTrack

URL: http://www.httrack.com/

15 http://hounder.org/features.html

11.1 Overview

HTTrack is designed to create mirrors from existing sites and not for indexing. A good tool for users unfamiliar with web crawling and that enjoy a good GUI.

11.2 Advantages

- HTTrack can follow links that are generated with basic JavaScript and inside Applets or Flash

11.3 Drawbacks

- No integration with indexing systems.

5.1.2.12 Hyper Estraier

URL: http://fallabs.com/hyperestraier/index.html

12.1 Overview

Hyper Estraier has some characteristics like high performance search and P2P support making it an interesting solution to add search to an existing website. The GNU Project is using Hyper Estraier to search its high number of docs making it a good solution when looking at collections approximately 8 thousands documents in size.

12.2 Advantages

- Useful to add search functionality to a site
- P2P support

12.3 Drawbacks

- Only one core developer

5.1.2.13 mnoGoSearch

URL: http://www.mnogosearch.org/

13.1 Overview

mnoGoSearch is a solution for a small enterprise appliance to add search ability to an existing site or intranet. The project is a bit outdated and due to the dependency on a specific vendor other solutions should be considered.

13.2 Advantages

- Used by MySQL

13.3 Drawbacks

- Little information about scalability and extensibility.
- Dependance on the vendor Lavtech for future development

5.1.2.14 Nutch

URL: http://nutch.apache.org/

14.1 Overview

Nutch is one of the most developed and active projects in the web crawling field. The need to scale and distribute the Nutch, lead to Doug Cutting, the project creator, started developing Hadoop - a framework for reliable, scalable and distributed computing.

This means that not only the project is developing itself but it also works with Hadoop, Lucene, Tika and Solr. The project is seeking to integrate other pieces of software such as HBase too (Bialecki 2009).

Another strong point for Nutch are the existing deployed systems with published case studies (M. Michael et al. 2007) and (J. E Moreira et al. 2007)

The biggest drawback in Nutch is the configuration and tuning process, combined with the need to understand how the crawler works to get the desired results. For large scale web crawling, Nutch is a stable and complete framework.

14.2 Advantages

- Nutch has a highly modular architecture allowing developers to create plugins for the following activities: media-type parsing, data retrieval, querying and clustering; (Khare et al. 2004)
- Nutch works under the Hadoop framework so it features cluster capabilities, distributed computation (using MapReduce) and a distributed filesystem (HDFS) if needed;
- Built in scalability and cost effectiveness in mind (Cafarella and Cutting 2004)
- Support to parse and index a diverse range of documents using Tika, a toolkit to detect

and extract metadata

- Integrated Creative Commons plugin;
- The ability to use other languages such as Python to script Nutch;
- There is an adaption of Nutch called NutchWAX (Nutch Web Archive eXtensions) allowing Nutch to open ARC files used by Heritrix;
- Top level Apache project, high level of expertise and visibility around the project.

14.3 Drawbacks

- Complexity in the framework
- The integrated MP3 parser is deprecated, based on “Java ID3 Tag Library” and did not work when tested in Nutch.

5.1.2.15 Open Search Server

URL: http://www.open-search-server.com/

15.1 Overview

Open Search Server is a good solution for small appliances. Unfortunately it is not well documented in terms of how extensible it is.

15.2 Advantages

- Quite easy to implement and set it running.

15.3 Minuses

- Dependence on the commercial component for development. Small community.
- Scarce documentation.
- Some problems handling special characters.
- There is little information on extending the software.

5.1.2.16 OpenWebSpider

URL: http://www.openwebspider.org/

16.1 Overview

This project might be interesting for those interested in the .NET framework, and in C# programming the intent to build a small to middle sized data collection.

16.2 Advantages

- MP3 support
- Crawl and database integration

16.3 Drawbacks

- Only one developer;
- Source disclosed but since no one else is working on the project and because there is no source code repository, it is not a real open source project;
- Mono Framework might constitute a problem for those concerned with patent issues;
- There is no proof of concept;
- Using relational database might not scale well.

5.1.2.17 Pavuk

URL: http://pavuk.sourceforge.net/

17.1 Overview

Pavuk is a complement to tools like Wget, still it does not offer indexing functionality.

17.2 Advantages

- Complement solutions like wget and HTTrack with filters for regular expressions and functions alike.

17.3 Drawbacks

- No development since 2007.
- No indexing features.

5.1.2.18 Sphider

URL: http://www.sphider.eu/

18.1 Overview

Sphider is a complete solution with crawler and web search that can run on a server just with PHP and MySQL. To add integrated searching functionality for existing web appliances might be a good solution with little requirements.

18.2 Advantages

- Easy to setup and integrate into an existing solution

18.3 Drawbacks

- The index is a relational database and might not scale well to millions of documents.

5.1.2.19 Xapian

URL: http://xapian.org/

19.1 Overview

Xapian is a search engine that relies on ht://Dig for crawling.

If a project has no problem in using CGI and relying on a outdated crawler, but rather puts the effort in having Linux distros packages, then this software can be an option.

19.2 Advantages

- Xapian “currently indexes over 50 million mail messages” in “gmane” lists proving that it can handle a connection at least that size
- Scaling to large document collections
- Still in active development
- Packages for some Linux distributions

19.3 Drawbacks

- The index can only be used by Xapian
- Using CGI
- Depends on ht://Dig for crawling

5.1.2.20 YaCy

URL: http://yacy.net/

20.1 Overview

For scientific projects like Sciencenet that have several machines across the world with different architectures it can be considered a good solution.

20.2 Advantages

- Yacy is a distributed search engine working like the P2P model. It is decentralized, even if one node goes down the search engine continues to work.
- It is easy to set Yacy working and it is quick to setup a P2P search network.

20.3 Drawbacks

- Hard to understand how customizable is outside the existing parameters.
- P2P search can be slow according to the Nutch FAQ (FAQ 2010)

5.1.3 Overview

From all the tools in analysis the ones with recent development reveal tend also to be the ones where scalability is a core issue. Tools like Bixo, Heritrix, Nutch and Yacy are designed to handle large data collections as the Web grows bigger.

According to each Web crawler tool functionalities and capabilities they can be placed in three categories:

- mirroring a collection with tools that don't do indexing but copy websites;
- medium collection crawling;

- large collection crawling.

It is important to note that the distinction between a medium and large collection is hard to make, here a large collection means near whole web crawl (more than 200 million documents) while medium means a subset from the Web (50 to 200 million documents). The differentiation between medium and large was made taking also into account the largest known deployed system (for each tool) because some tools declared to have the ability to large web crawl but caressed a proof of concept.

Mirroring a collection

- GNU Wget, Heritrix, HTTrack, Pavuk

Medium collection

- Aspseek, crawler4j, DataparkSearch, Ebot, GRUB, Hounder, ht://Dig, Hyper Estraier, mnoGoSearch, Open Search Server, OpenWebSpider, Sphider, Xapian

Large collection

- Bixo, Nutch and Yacy

5.1.4 Conclusion

For the most cases where only one enterprise intranet or a small specific subset of the web needs to be processed, lighter and with faster configuration tools might be enough. In this case solutions in the medium collection category, with the help from Table 3 and Table 4 in order to choose the programming language and indexing system preferred, can constitute a good choice.

When looking for solutions for large collections, YaCy with a P2P framework is an option. This is an interesting software when speed is not crucial, focus goes into a distributed architecture and into an easy setup.

To provide reliable, fast and scalable computing Bixo and Nutch are the best answer. This is supported in part because both rely on Hadoop, an industry wide adopted and proven framework, with several success cases such as Yahoo clusters (YAHOO, 2008) (YAHOO, 2010), (Shvachko, Kuang, Radia, and Chansler 2010), Facebook (FACEBOOK, 2010)

Last.fm (Dittus 2008) and Spotify using Hadoop (Bernhardsson 2009) (Kreitz and Niemela 2010). These are just a few examples of organizations using Hadoop. The list is however much more comprehensive (PoweredBy 2010).

The main difference between them is that Bixo relies on Cascading to complete the workflow and does not do indexing while Nutch indexes using Lucene.

In general, solutions using the Lucene index tend to have fast retrieval times and requiring few space on disk (good characteristics for a search engine) in comparison to other solutions (Middleton and Baeza-yates).

If the choice has to be made between Bixo and Nutch, it depends on the goal to integrate an existing system and workflow in order to do data mining or related jobs, choose Bixo. If it is to build a system with a search engine to handle a massive document collection, Nutch is the tool of choice (TutorialSIGIR 2009) (Singh Singh 2009 2009).

6 Methodology

The methodology used in this study consists in a case-based approach. First, an overview of the essential components for music recommendation systems were identified and studied.

Then, using a bottom-down approach to assess what was the most urgent component that needed to be solved, led to understand that the first area of interest to study and develop was indexing, because there was no index on which recommendation or other research could work with.

To create this index of music, it was necessary to analyze the state of the art of the available open source tools for Web crawling. After this stage, further testing was conducted in order to understand which were the most promising open source tools to address the problem of this work. This study led to conclude which was the most appropriate tool for the development of the system prototype.

While choosing the appropriate tool, the selected methodology also took into consideration the opinions of the community around each open source project. The involved community it is important because, for example, with the evolving HTML specification and the constant rise of new and updated file formats, like the new Microsoft Office formats or the addition of a new version of ID3 tags to MP3, means that it is relevant to open source projects to have an active development community around them that can really develop the projects to catch up with new demands.

After the exploratory study a project implementation phase took place and for the prototype it was used the Timebox methodology (Jalote, Palit, and Kurien 2004) due to the time constraints.

7 Proposal

This work proposes to create a scalable MP3 index and search engine. To create the index, a given set of base URLs from websites containing MP3s, mostly netlabels, will be fed into the crawler that will crawl the web pages scorching for MP3s. After this, then parsing will occur looking for relevant metadata to index. The relevant metadata that was considered is the following:

- the artist of the track;
- the title of the track;
- the album of the track.

This is the most essential data for almost every user. If the genre and the year are available, they will also be added to the index. Further metadata like comments and the album artwork may also be indexed.

In order to accomplish the goal of building a scalable index with a web search engine for music, the decision for the tool of choice fell into Nutch.

Nutch is an open search Web search program that can run on a single machine, however the true value in terms of scalability comes from running in a Hadoop cluster. This allows Hadoop to scale well adding new machines to meet new demands in terms of storage or processing power. In terms of search this is a big plus because complex tasks can run in a large cluster distributing processing jobs across machines.

In terms of functionality Nutch is a comprehensive tool that is composed by a crawler, a set of parsers, the capability to build a link-graph structure and also provides web search.

The indexing component is done through the integration with Lucene, a Java text search engine. Since the resulting index can be quite large, in terms of disk space usage, Nutch works with HDFS (Hadoop Distributed File System) to distribute large amounts of data across cluster nodes.

Once Nutch is built on top of Lucene, it has some advantages brought by the Lucene framework:

- OS agnostic – because it is based on Java, Lucene can run on multiple platforms and operative systems
- Offers an API for other languages – Lucene can be read and used through existing

bindings for diverse languages such as Delphi, Perl, C#, C++, Python, Ruby and PHP.

- The index is independent from the framework - can be used with other tools or APIs. This means the created index is not dependent solely on the Nutch framework, it can be accessed by other applications.

Another visible advantage of Nutch using Lucene is that the search component can also be achieved through Solr, an enterprise search platform.

Besides being a complete solution with all the parts needed, Nutch also offers a modular architecture where each part can be extended and customized with greater ease. This can be attained through the plugin structure where each plugin can be extended or changed to work according to the needed specification. Everything from indexing, parsing and querying can be changed or extended since all these steps are done by plugins. With the plugin system Nutch offers greater extensibility, flexibility and maintainability.

Every search engine has its own way of ranking results for a determined query. With Nutch this is no exception but it also has the ability to explain and understand results and why they were scored that way. On top of that it is easy to boost indexed fields, including custom fields added by researchers, through the configuration of a simple XML file.

Nutch was the selected tool because the project is still having rapid development with new features and integrations being added that could be of value for future works. Such improvements include the integration with HBase and greater integration with Solr.

It is frequent to see developers from the community, outside the development team, contributing back to Nutch with patches to fix or improve the system.

Having understood the Nutch plugin framework the next step is to extend the existing parser plugins to index MP3.

After the implementation of the prototype,assessment concerning the obtained results will be carried out.

To conclude the document, the taken approach and chosen solutions will be revised, in order to evaluate the work done.

8 Validation & Assessment

In this section, an analysis of all the necessary steps to create a scalable index and web search engine for music on the Internet is made. Starting from the simple step of collecting all the seed URLs (see section 8.1), then how to crawl and indexing these URLs (section 8.3), followed by the parsing and indexing of MP3 (section 8.4). After this, it will be introduced the search with Nutch (section 8.7), in which it is important to understand how the boosting fields in a search engine ranking system work and finally the integration with Solr is also taken into account (section 8.9).

At the end of this some references to the chosen project management methodology are also made.

8.1 Gathering Seed URLs

Seed URLs correspond to the set of pages from which the crawler starts gathering links to other pages. In fact, these seed URLs are responsible for kickstarting the crawling process.

In this work, because there is no interest in crawling the entire Web, the crawler is limited to the domains available in the seed URLs. So the seed URL list must contain all the domains for desired crawling process.

The goal is to collect only in netlabels and pages where Creative Common content is available. In order to do this a small set of URLs were firstly collected for some of the Portuguese netlabels.

In addition to this to this small set of Portuguese netlabels others were also added from the global Netlabel Index[16].

In order to complete the list of netlabels, a page containing a comprehensive list from netlabels[17] was processed through a technique called "web scraping". Web scraping is a technique used to extract data from a website.

Web scraping was done by building a small script in the Python language with the assistance of the parser "Beautiful Soup" to extract all the links to the netlabels in the page inside a specific table. The source code in this is presented in the annex section 11.1.

16 http://www.netlabelindex.com/browse_label.php Accessed: 2010-09-20
17 http://www.clongclongmoo.org/system/index.php?cat=00_netaudio&page=05_netlabellist Accessed: 2010-09-20

8.2 Configuring Nutch

The Nutch tool is configured through the "nutch-site.xml" file. This file overrides the "nutch-default.xml" file that contains the Nutch tool default configurations.

In order to tune up Nutch it is recommended to see what is proposed in the default configuration file. Each of the properties in the file contains a detailed description that can be quite useful in understanding its specific behavior.

8.3 Crawl and Indexing

The crawling and indexing processes are performed through a series of different steps. The process starts from a set of given base URLs, called "seed URLs", which Nutch can crawl through the HTML structure in order to find links to MP3 files that are indexed at the end. The entire process, depicted in Figure 8.1, can be detailed in the following steps:

- Inject – this corresponds to the first step, where URLs are injected into the crawl database.

After this, crawling and indexing are performed through a generate/fetch/update cycle that is repeated according to the desired crawling depth (named as "depth" in Figure 8.1). The description of this processis the following:

- Generate – this is the step in which a set of URLs are selected to be fetch;
- Fetch – the actual URLs selected previously are fetched;
- Update – this step updates the crawling database with the results from the fetch step.

This generate/fetch/update steps are repeated (counted by the variable named "n" in Figure 8.1) until all the cycles are complete (when "depth = n"). The following indexing is executed afterwards:

- Index – this is the step where the index is created from the different segments created in the previous steps.

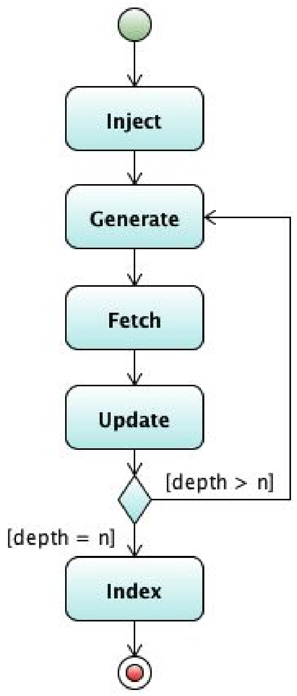


Figure 8.1: The crawling and indexing process

At the end of all these steps a directory called “indexes” is created containing the index that can be used by the Nutch search engine.

All the steps can be run altogether with the Nutch crawling command, or separately for more control over the crawling and indexing process.

The Nutch tool property “db.ignore.external.links” was set to true, so that external hosts different from the ones initially inserted in the seed URLs would be ignored. The effect of this configuration is that the crawling process gets limited to the domains injected in the seed URLs list.

8.4 Parsing and Indexing MP3

The following sections will provide some more details about the MP3 files parsing and indexing processes. This section starts by explaining how it is possible to extend the Nutch capabilities through the usage of specific indexing plugins. Then a short explanation on how the index plugin is built is also provided.

8.4.1 Extending the index plugin

In order to add to Nutch the capability to index MP3 ID3 tag fields in MP3 files the approach was to create a parsing plugin capable of extending the existing parse-tika plugin.

Nutch integrated Tika, which is an Apache Foundation project of a “toolkit for detecting and extracting metadata and structured text content from various documents using existing parser libraries”, in order to provide better support for a diverse range of document formats.

The steps involved in the creation of a plugin include writing two XML files: one that describes the plugin (plugin.xml, annex section plugin.xml) and the other to build the plugin (build.xml, annex section 11.3.3).

In terms of code implementation this was achieved using the Nutch extension point for indexing, called “IndexingFilter”. This permits one to add metadata to the indexed fields and the Tika Interface XMPDM[18] was possible to parse and index fields like “Artist”, “Title” and “Album”. For more information on the implementation see the attachments under Mp3IndexingFilter.java in 11.3.2 .

After this it is necessary to assure that the mime types are correct in the Tika file “tika-mimetypes.xml”, so that the extension “.mp3” is handled according to the content with the mime type “audio/mpeg”.

8.4.2 Building the index plugin

The Nutch plugins can be built altogether by running “ant plugins”. Ant is a command line utility used mainly to build Java applications. To add the created plugin to the build process the “build-plugins.xml” file must be changed. To deploy the plugin after the target deployment the corresponding Ant line must be added:

```
<target name="deploy">
    <ant dir="index-mp3" target="deploy"/>
```

Finally the newly created indexer called “index-mp3” is added to the “plugin.includes” property in the “nutch-site.xml” file.

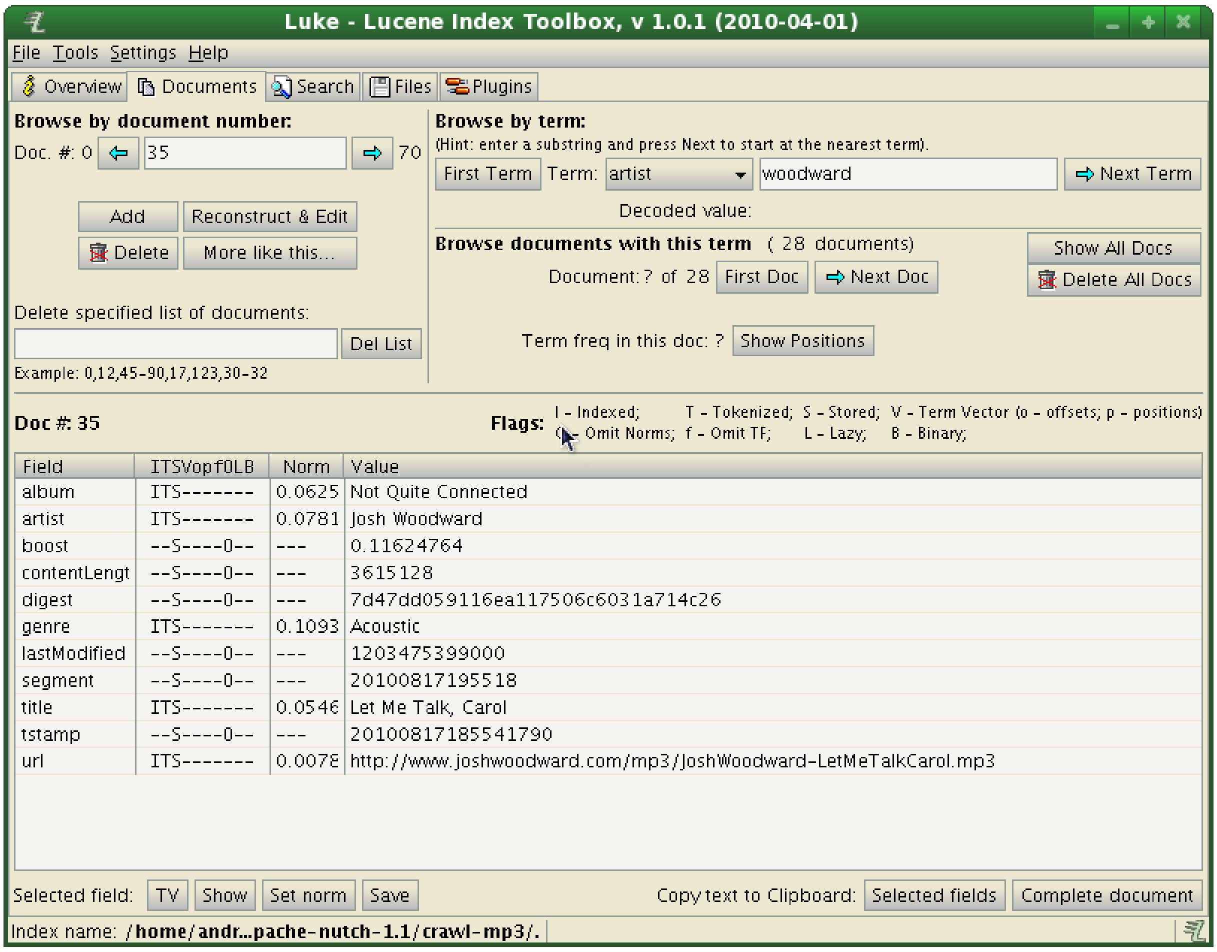
18 http://tika.apache.org/0.7/api/org/apache/tika/metadata/XMPDM.html Accessed: 2010-09-21

8.5 Speeding up the fetch

In order to speed up the crawling process only 1048576 bytes or 1 megabyte from all files is downloaded. For MP3 files this is well enough to capture all the file headers, once these headers are located at the beginning of the file. This is specified in the property “file.content.limit” in the “nutch-site.xml” file.

8.6 Browsing the index

To make sure that the correct fields and respective values had been correctly added to the index, a useful tool named “Luke” was used to navigate and query the index, assuring that the MP3s files were parsed and indexed correctly. In Figure 8.2 it is possible to see the custom fields like "artist, genre, album".


Luke - Lucene Index Toolbox, v 1.0.1 (2010-04-01)
File Tools Settings Help
Overview Documents Search Files Plugins
Browse by document number:
Add Reconstruct & Edit
Delete More like this...
Delete specified list of documents:
Del List
Browse by term:
First Term
Next Term
Decoded value:
Show All Docs
Delete All Docs
First Doc Next Doc
Show Positions
Selected field: TV Show Set norm Save
Copy text to Clipboard: Selected fields Complete document

Doc. #: 0 35 70

(Hint: enter a substring and press Next to start at the nearest term).

Term: artist woodward

Browse documents with this term (28 documents)

Document: ? of 28

Term freq in this doc: ?

Example: 0,12,45-90,17,123,30-32

Doc #: 35

Flags: I - Indexed; T - Tokenized; S - Stored; V - Term Vector (o - offsets; p - positions) O - Omit Norms; f - Omit TF; L - Lazy; B - Binary;

Field	ITSVopfOLB	Norm	Value
album	ITS-------	0.0625	Not Quite Connected
artist	ITS-------	0.0781	Josh Woodward
boost	--S----0--	---	0.11624764
contentLengt	--S----0--	---	3615128
digest	--S----0--	---	7d47dd059116ea117506c6031a714c26
genre	ITS-------	0.1093	Acoustic
lastModified	--S----0--	---	1203475399000
segment	--S----0--	---	20100817195518
title	ITS-------	0.0546	Let Me Talk, Carol
tstamp	--S----0--	---	20100817185541790
url	ITS-------	0.007ε	http://www.joshwoodward.com/mp3/JoshWoodward-LetMeTalkCarol.mp3

Index name: /home/andr...pache-nutch-1.1/crawl-mp3/.

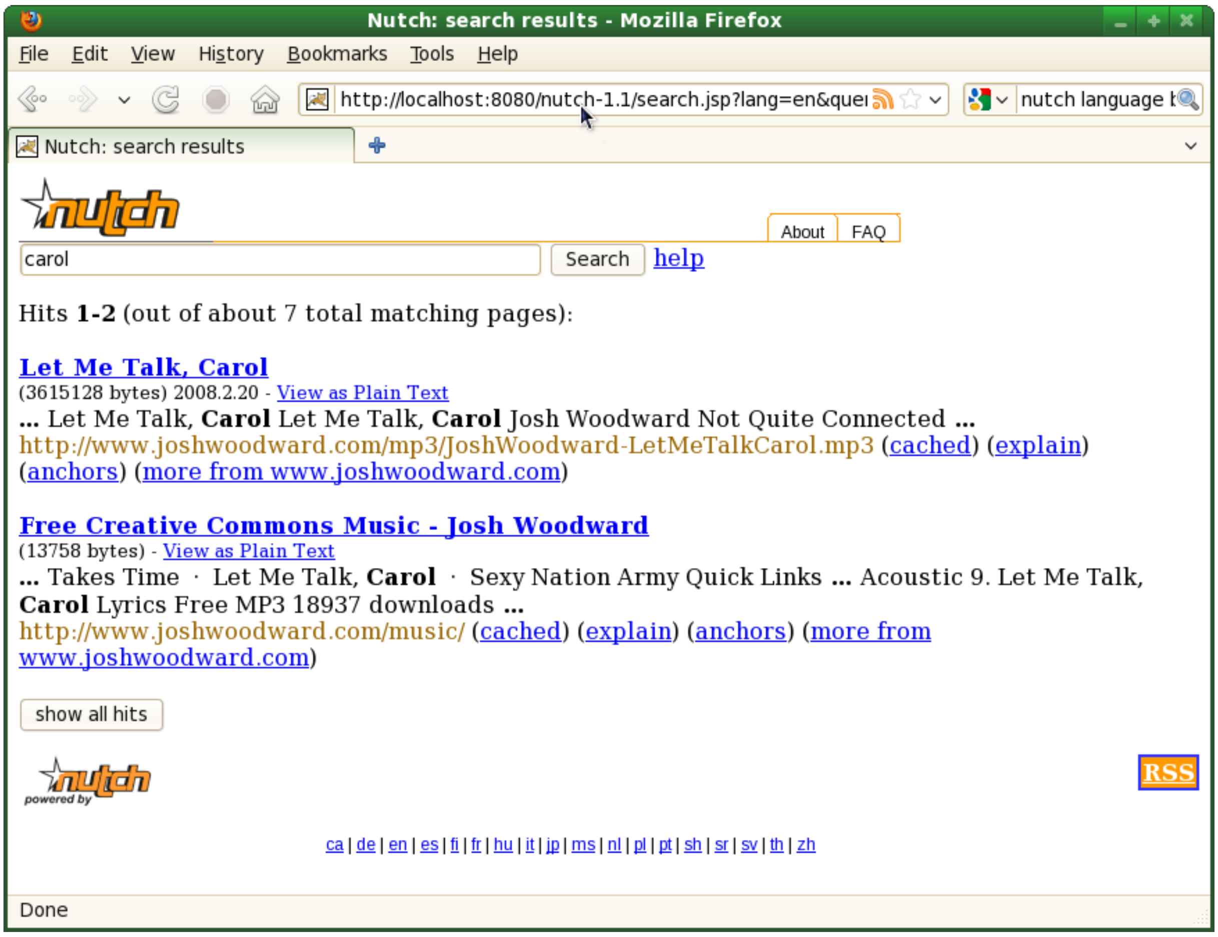
Figure 8.2: Screenshot from Luke showing a MP3 document

8.7 *Searching with Nutch*

Performing searches with Nutch can be setup through the usage of a Tomcat server, or any other application server supporting servlets like Resin or Jetty.

First, it is necessary to deploy the “nutch.war” file running “ant war” in the command line and then to deploy it in Tomcat. With the Nutch servlet in place the typical search interface is presented in the browser.

In Figure 8.3 it is represented a search and the respective results by MP3 files containing “carol” in the document fields. HTML results are also displayed in conjunction with MP3 results because, for this purpose of showing sample results, they do not constitute a problem.


Nutch: search results - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/nutch-1.1/search.jsp?lang=en&quer
nutch language
Nutch: search results
About FAQ
carol
Search help
show all hits
RSS
powered by
ca | de | en | es | fi | fr | hu | it | jp | ms | nl | pl | pt | sh | sr | sv | th | zh
Done

Hits **1-2** (out of about 7 total matching pages):

Let Me Talk, Carol
(3615128 bytes) 2008.2.20 - View as Plain Text
... Let Me Talk, **Carol** Let Me Talk, **Carol** Josh Woodward Not Quite Connected **...**
http://www.joshwoodward.com/mp3/JoshWoodward-LetMeTalkCarol.mp3 (cached) (explain) (anchors) (more from www.joshwoodward.com)

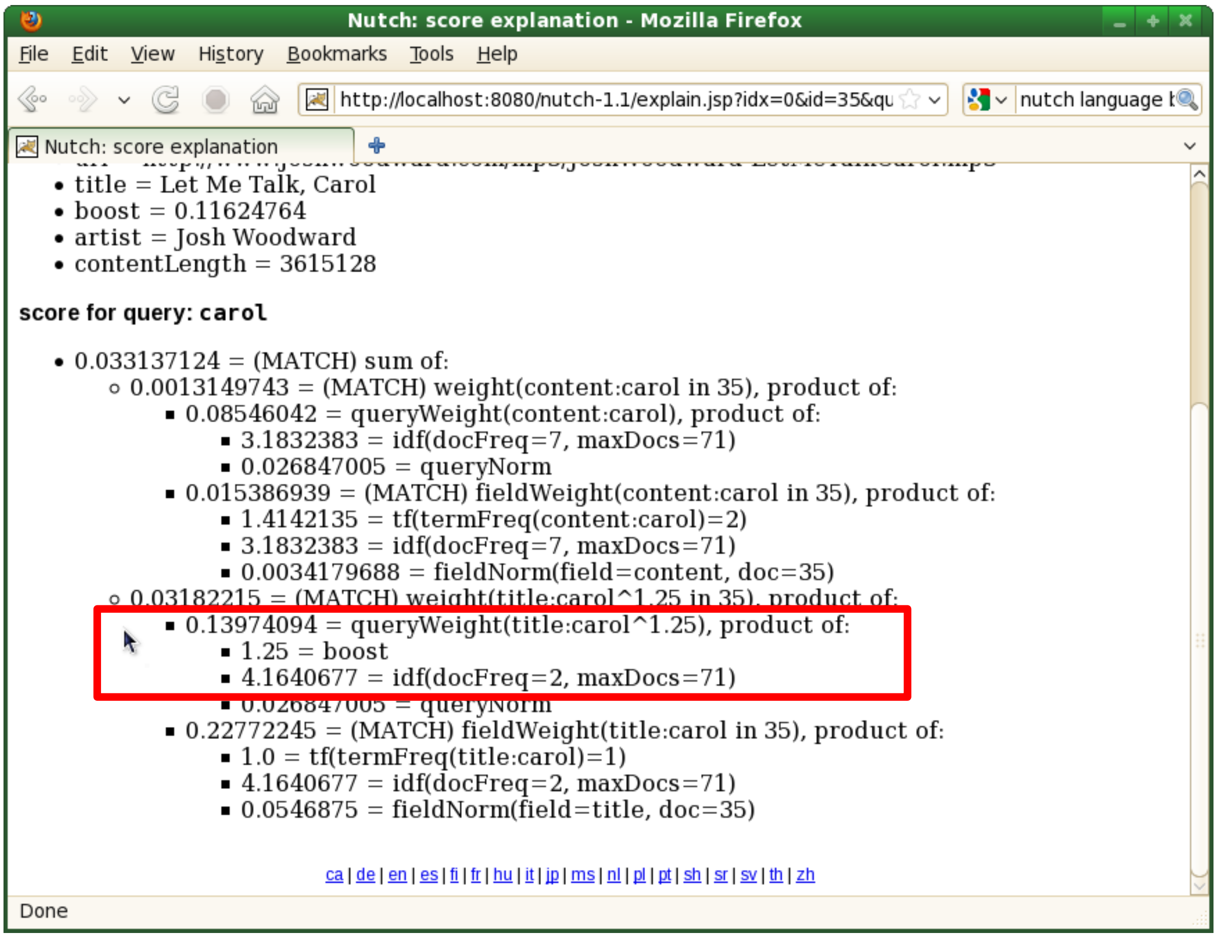
Free Creative Commons Music - Josh Woodward
(13758 bytes) - View as Plain Text
... Takes Time · Let Me Talk, **Carol** · Sexy Nation Army Quick Links **...** Acoustic 9. Let Me Talk, **Carol** Lyrics Free MP3 18937 downloads **...**
http://www.joshwoodward.com/music/ (cached) (explain) (anchors) (more from www.joshwoodward.com)

Figure 8.3: Search in Nutch by "carol"

8.8 Boosting fields in the search engine ranking system

Every search engine has its own way of ranking results according to a determined query. Nutch is no exception. The major advantage in comparison to other search engines is the possibility to explain why a result was ranked and to meddle with boost values to reorder the results.

In Figure 8.4 there is an explanation provided by Nutch (under the "explain" link) for the first result in Figure 8.3, what is important to notice from the formula is the 1.25 value (highlighted in the a red rectangle in Figure 8.4) representing the boost value for the title field. With this 1.25 boost in the title field, the MP3 file is the first result in Figure 8.3.


Nutch: score explanation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/nutch-1.1/explain.jsp?idx=0&id=35&qu
nutch language
Nutch: score explanation
Done

- title = Let Me Talk, Carol
- boost = 0.11624764
- artist = Josh Woodward
- contentLength = 3615128

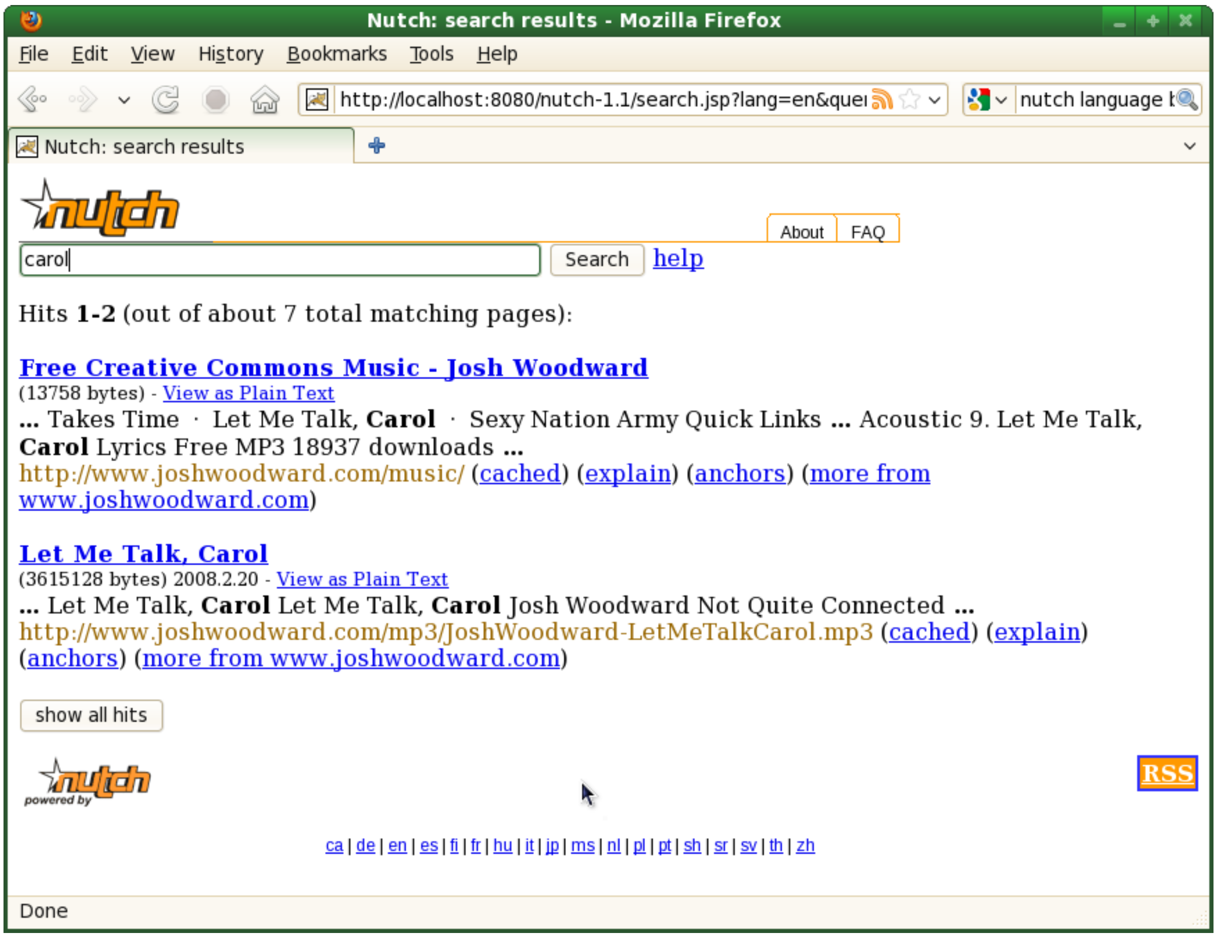
score for query: carol

```
• 0.033137124 = (MATCH) sum of:
   ◦ 0.0013149743 = (MATCH) weight(content:carol in 35), product of:
      ▪ 0.08546042 = queryWeight(content:carol), product of:
         ▪ 3.1832383 = idf(docFreq=7, maxDocs=71)
         ▪ 0.026847005 = queryNorm
      ▪ 0.015386939 = (MATCH) fieldWeight(content:carol in 35), product of:
         ▪ 1.4142135 = tf(termFreq(content:carol)=2)
         ▪ 3.1832383 = idf(docFreq=7, maxDocs=71)
         ▪ 0.0034179688 = fieldNorm(field=content, doc=35)
   ◦ 0.03182215 = (MATCH) weight(title:carol^1.25 in 35), product of:
      ▪ 0.13974094 = queryWeight(title:carol^1.25), product of:
         ▪ 1.25 = boost
         ▪ 4.1640677 = idf(docFreq=2, maxDocs=71)
         ▪ 0.026847005 = queryNorm
      ▪ 0.22772245 = (MATCH) fieldWeight(title:carol in 35), product of:
         ▪ 1.0 = tf(termFreq(title:carol)=1)
         ▪ 4.1640677 = idf(docFreq=2, maxDocs=71)
         ▪ 0.0546875 = fieldNorm(field=title, doc=35)
```

ca | de | en | es | fi | fr | hu | it | jp | ms | nl | pl | pt | sh | sr | sv | th | zh

Figure 8.4: Explanation on the score attributed to the first result in Figure 8.3

To reorder the results the configuration in the "nutch-site.xml" file must be changed. Now the property "query.title.boost" will be set to 0.25 giving the "title" field a 0.25 value boost. The result is that for the same query "carol" the MP3 document now is second in the results list as Figure 8.5 illustrates.


Nutch: search results - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/nutch-1.1/search.jsp?lang=en&quer
nutch language
Nutch: search results
About
FAQ
Search
help
show all hits
RSS
ca | de | en | es | fi | fr | hu | it | jp | ms | nl | pl | pt | sh | sr | sv | th | zh
Done

carol

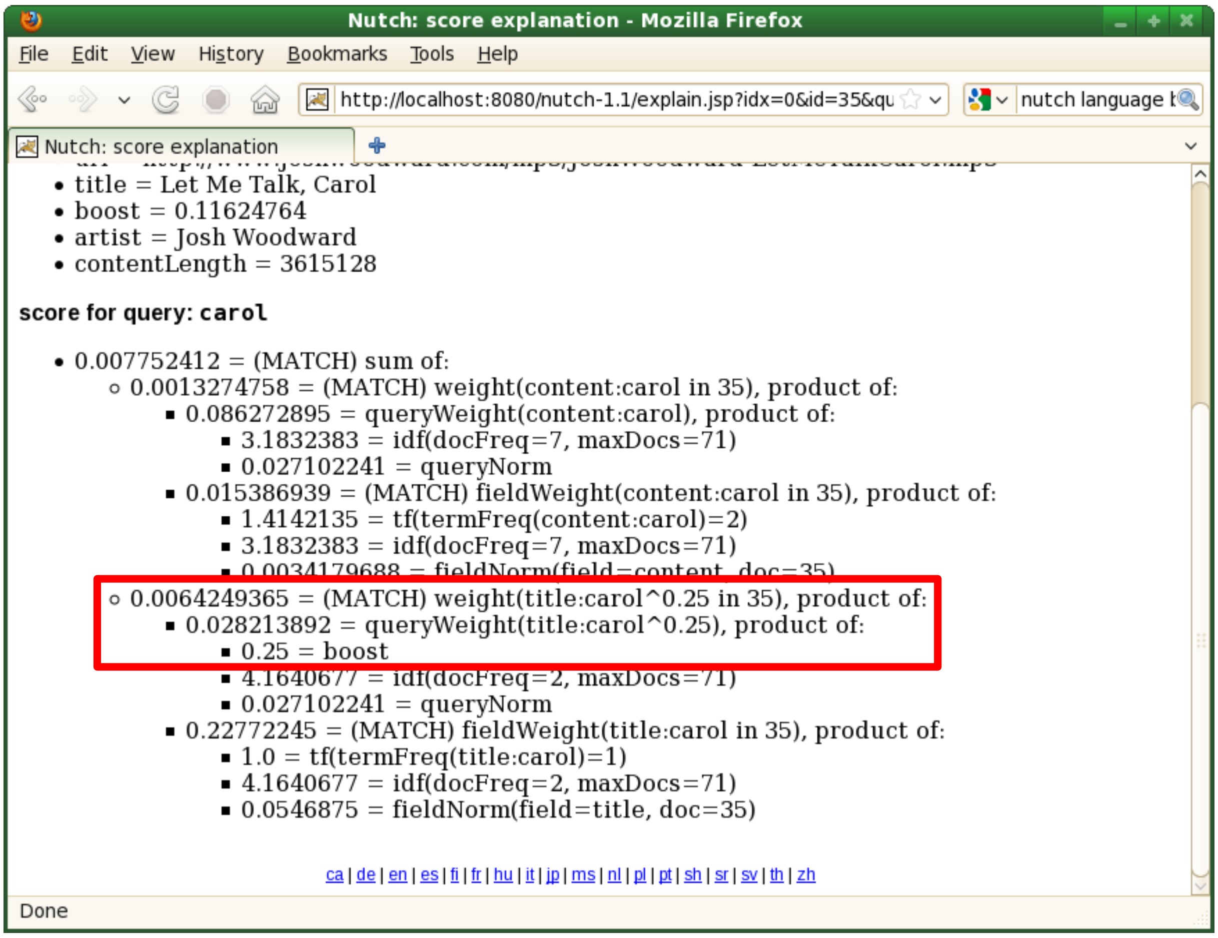
Hits **1-2** (out of about 7 total matching pages):

Free Creative Commons Music - Josh Woodward
(13758 bytes) - View as Plain Text
... Takes Time · Let Me Talk, **Carol** · Sexy Nation Army Quick Links ... Acoustic 9. Let Me Talk, **Carol** Lyrics Free MP3 18937 downloads ...
http://www.joshwoodward.com/music/ (cached) (explain) (anchors) (more from www.joshwoodward.com)

Let Me Talk, Carol
(3615128 bytes) 2008.2.20 - View as Plain Text
... Let Me Talk, **Carol** Let Me Talk, **Carol** Josh Woodward Not Quite Connected ...
http://www.joshwoodward.com/mp3/JoshWoodward-LetMeTalkCarol.mp3 (cached) (explain) (anchors) (more from www.joshwoodward.com)

Figure 8.5: Search in Nutch by "carol", with different ranking order from Figure 8.3

To understand what happened in Figure 8.5, in order to place the MP3 result as the second result, attending to Figure 8.6 it is possible to see that now the “title” field boost is only 0.25 (also highlighted in a red rectangle). The effect of this is that now the MP3 result has a lower score and is surpassed by other documents in the ranking.


Nutch: score explanation - Mozilla Firefox

http://localhost:8080/nutch-1.1/explain.jsp?idx=0&id=35&qu

```
• title = Let Me Talk, Carol
• boost = 0.11624764
• artist = Josh Woodward
• contentLength = 3615128

score for query: carol

• 0.007752412 = (MATCH) sum of:
    ◦ 0.0013274758 = (MATCH) weight(content:carol in 35), product of:
        ▪ 0.086272895 = queryWeight(content:carol), product of:
            ▪ 3.1832383 = idf(docFreq=7, maxDocs=71)
            ▪ 0.027102241 = queryNorm
        ▪ 0.015386939 = (MATCH) fieldWeight(content:carol in 35), product of:
            ▪ 1.4142135 = tf(termFreq(content:carol)=2)
            ▪ 3.1832383 = idf(docFreq=7, maxDocs=71)
            ▪ 0.0034179688 = fieldNorm(field=content, doc=35)
    ◦ 0.0064249365 = (MATCH) weight(title:carol^0.25 in 35), product of:
        ▪ 0.028213892 = queryWeight(title:carol^0.25), product of:
            ▪ 0.25 = boost
            ▪ 4.1640677 = idf(docFreq=2, maxDocs=71)
            ▪ 0.027102241 = queryNorm
        ▪ 0.22772245 = (MATCH) fieldWeight(title:carol in 35), product of:
            ▪ 1.0 = tf(termFreq(title:carol)=1)
            ▪ 4.1640677 = idf(docFreq=2, maxDocs=71)
            ▪ 0.0546875 = fieldNorm(field=title, doc=35)

ca | de | en | es | fi | fr | hu | it | jp | ms | nl | pl | pt | sh | sr | sv | th | zh
```

Figure 8.6: Explanation on the score attributed to the second result for the query "carol"

8.9 Integration with Solr

Solr is an open source enterprise search platform, by adding Solr integration, Nutch takes advantage of Solr features such as:

- queries – possibility to make more complex queries than the ones in Nutch
- highlighting – within the results relevant parts and keywords are highlighted
- faceted search - dynamic clustering of items or search results into categories, for example to search for an artist and get albums as aggregated results
- sorting – sort query results

In order to use these features the needed configuration steps are explained in section 8.9.1. Thereupon queries can be made using Solr, some example queries to demonstrate the advantages are shown in section 8.9.2.

8.9.1 Configuration

In order to use Solr with the Nutch index some mapping between the existing fields in Nutch and Solr must be performed to index them correctly. This is achieved by adding the new fields to the "schema.xml" file in Solr.:

```
  <field name="artist" type="string" stored="true" indexed="true"/>
  <field name="album" type="string" stored="true" indexed="true"/>
  <field name="genre" type="string" stored="true" indexed="true"/>
  <field name="cc" multiValued="true" type="string" stored="true"
indexed="true"/>
```

After mapping the fields correctly in the "schema.xml" file all there is to do is to run Nutch with the "solrindex" option to index it in Solr:

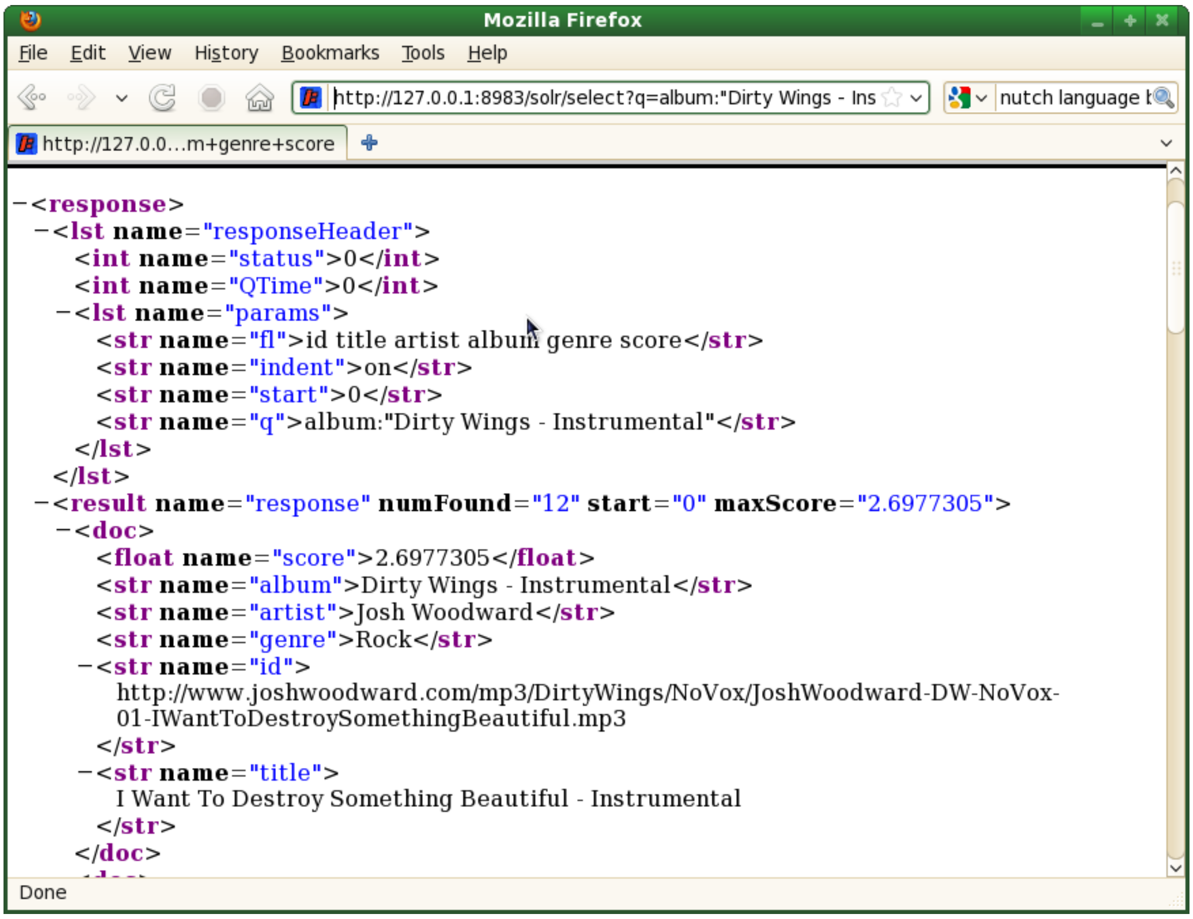
```
$ bin/nutch solrindex http://127.0.0.1:8983/solr/ crawl-
mp3/crawldb crawl-mp3/linkdb crawl-mp3/segments/*
```

8.9.2 Queries

With the Solr tool running and all the fields indexed, using any browser, it is possible to start making queries.

A good example is presented in Figure 8.7, where it is made a query (parameter "q") by the field "album" with the value "Dirty Wings – Instrumental" (queries in Solr can have the format [FIELD]:[VALUE]). As a result of the previous query all twelve songs from the specified album (numFound="12") are retrieved. Also the illustrated query uses a filter to limit the amount of information in the response (parameter "fl") showing only the fields id, title, artist, album, genre and score. The complete URL used: "http://127.0.0.1:8983/solr/select?q=album:%22Dirty%20Wings%20-%20Instrumental%22&start=0&indent=on&fl=id+title+artist+album+genre+score".

This type of query is more user friendly, because there is no SQL involved. Just by knowing the fields to look for, Solr can return all the desired documents in XML format.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1:8983/solr/select?q=album:"Dirty Wings - Ins
nutch language
http://127.0.0...m+genre+score
Done

```
-<response>
 -<lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">0</int>
  -<lst name="params">
      <str name="fl">id title artist album genre score</str>
      <str name="indent">on</str>
      <str name="start">0</str>
      <str name="q">album:"Dirty Wings - Instrumental"</str>
    </lst>
  </lst>
 -<result name="response" numFound="12" start="0" maxScore="2.6977305">
  -<doc>
      <float name="score">2.6977305</float>
      <str name="album">Dirty Wings - Instrumental</str>
      <str name="artist">Josh Woodward</str>
      <str name="genre">Rock</str>
    -<str name="id">
        http://www.joshwoodward.com/mp3/DirtyWings/NoVox/JoshWoodward-DW-NoVox-
        01-IWantToDestroySomethingBeautiful.mp3
      </str>
    -<str name="title">
        I Want To Destroy Something Beautiful - Instrumental
      </str>
    </doc>
```

Figure 8.7: Query by album name using Solr with response in XML

8.10 Project Management

The TimeBox was chosen as the project management methodology because it has proven to be the an adequate choice and it has allowed to focus on core functionality in a limited time frame. As a result the most essential functionalities were delivered on time. This is a great advantage in the perspective that by the end of the project there is a working prototype instead of fragmented pieces of software.

9 Conclusions

The Internet, in conjunction with the Creative Commons and Public Domain licenses, have made possible the access to a vast collection of freely available music. But without an index or database it is difficult to browse this sparse, wide and diverse collection. To perform searches in such a fragmented data amassment is practically impossible.

To address this problem, a case based study on existing music recommendation systems, with the necessary characteristics to solve this problem, was conducted. This case based study allowed the identification of several core components in music recommendation systems.

Since the missing database or index was impairing future work, it was definitely the most prominent issue to work on and it was addressed by these thesis. An analysis of the State of the Art of open source tools for Web crawling and indexing was assembled to understand what best software framework could directly tackle the problem.

From this study, a software piece called Nutch (Cafarella and Cutting 2004) was chosen for the prototype development. The decision was based on the software ability to scale well, to crawl, to process and index large amounts of information (M. Michael, J. E Moreira, D. Shiloach, and Wisniewski 2007) (José E. Moreira et al. 2007). It was also important the Nutch plugin system, as it assured a greater extensibility and flexibility (Khare, Cutting, Sitaker, and Rifkin 2004).

The prototype was built according to the TimeBox project management methodology, key functionalities were chosen and implemented. As a result of this methodology, at the end there was a working piece of software with the key features deployed.

To kick-start the crawling, a Python script was written to parse an existing web page containing a list of netlabels. This was helpful to create the initial seed list.

Crawling the web was done using the Nutch crawl script. This crawl script calls the inject step to insert seed URLs (the initial URLs to kickstart the crawling process), then loops a generate/fetch/update cycle that runs as many times as the desired crawl depth, and finishes with the index creation. Through the fine tuning of each of the steps instead of running them as a whole, probably, even more documents could be found in the websites. Results may vary as every site has its own structure, and a configuration optimal to crawl one site might not be the best for another one.

To the search engine and web crawler Nutch were added MP3 parsing and indexing capabilities, through the plugin system. This way, information present in the ID3 tags like artist, album or genre were added, making possible via the Web search engine to look for artists or music genres. On top of these, with Solr integration, more complex and specific queries could be made with the results being returned in the XML format.

Being able to crawl and index MP3 files using Nutch, a proven scalable software framework, attested that it is possible to build a large scale index for the MP3 collection available on the Internet, using Open Source software.

For future work, there is the will to evolve the prototype, so that it can accommodate new features. An important step would be to associate and index the MP3 files with the respective CC license. This could be done either by parsing the MP3 or the HTML from were the MP3 was found to look for a CC license.

Equally interesting is the ability to parse ZIP and Flash content in order to discover and extract MP3s contained in “zip” files or linked inside “swf” files like online Flash MP3 players. This feature would really boost the available collection size, as a large quantity of the music available is behind these two types of files.

With the index problem solved, it would be interesting to move future research into music recommendation, because, with the sparse and wide collection that is offered, becomes very difficult for a user without references to explore and find music enjoyable or convenient to the context in cause. “How to explore music on the Internet without musical or cultural references?” would be a possible question to address.

But even if an answer can be drawn to the previous question it is important to put the question: “Is there interest by Internet users in exploring the freely available collection of music online?”

These are just some examples of interesting topics for further research that can be perfectly supported by the infrastructure created in the current thesis.

This work help was done has been done in the hope that it will help to bootstrap research around freely available music on the Internet, for the benefit of research and Creative Commons music.

10 Bibliography

13. Internet Archive ARC files. [cited 29 September 2010]. Available from world wide web: <http://crawler.archive.org/articles/developer_manual/arcs.html>.

Creative Commons — Attribution 2.5 Portugal. [cited 17 September 2010]. Available from world wide web: <http://creativecommons.org/licenses/by/2.5/pt/>.

Ebot | Matteo Redaelli. [cited 21 September 2010]. Available from world wide web: <http://www.redaelli.org/matteo-blog/projects/ebot/>.

FAQ - Nutch Wiki. [cited 17 September 2010]. Available from world wide web: <http://wiki.apache.org/nutch/FAQ#Will_Nutch_be_a_distributed.2C_P2P-based_search_engine.3F>.

Gracenote | iTunes. [cited 27 September 2010]. Available from world wide web: <http://www.gracenote.com/casestudies/itunes/>.

Grooveshark Mobile Music. [cited 27 September 2010]. Available from world wide web: <http://m.grooveshark.com/>.

HDFS: Facebook has the world's largest Hadoop cluster! [cited 25 September 2010]. Available from world wide web: <http://hadoopblog.blogspot.com/2010/05/facebook-has-worlds-largest-hadoop.html>.

Hive/PoweredBy - Hadoop Wiki. [cited 26 September 2010]. Available from world wide web: <http://wiki.apache.org/hadoop/Hive/PoweredBy>.

Metrics - CC Wiki. [cited 25 September 2010]. Available from world wide web: <http://wiki.creativecommons.org/Metrics>.

Netlabel - Wikipedia, the free encyclopedia. [cited 17 September 2010]. Available from world wide web: <http://en.wikipedia.org/wiki/Netaudio>.

PoweredBy - Hadoop Wiki. [cited 27 September 2010]. Available from world wide web: <http://wiki.apache.org/hadoop/PoweredBy>.

Public Terabyte Dataset Project « Elastic Web Mining | Bixo Labs. [cited 25 September 2010]. Available from world wide web: <http://bixolabs.com/datasets/public-terabyte-dataset-project/>.

San Francisco Bay Area ACM , Archive » DM SIG – ACM Silicon Valley Data Mining Camp on November 1, 2009. [cited 25 September 2010]. Available from world wide web: <http://www.sfbayacm.org/?p=894>.

Scalability of the Hadoop Distributed File System (Yahoo! Hadoop Blog). [cited 25 September 2010]. Available from world wide web: <http://developer.yahoo.net/blogs/hadoop/2010/05/scalability_of_the_hadoop_dist.html>.

Tutorial - T6: IR Prototypes and Web Search Hacks with Open Source Tools | SIGIR'09. [cited 17 September 2010]. Available from world wide web: <http://sigir2009.org/Program/tutorials/T6>.

Yahoo! Launches World's Largest Hadoop Production Application (Yahoo! Hadoop Blog). [cited 25 September 2010]. Available from world wide web: <http://developer.yahoo.net/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html>.

Yahoo! Launches World's Largest Hadoop Production Application (Yahoo! Hadoop Blog). [cited 25 September 2010]. Available from world wide web: <http://developer.yahoo.net/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html>.

Aucouturier, J. -J, and F. Pachet. Finding songs that sound the same. In *Proceedings of IEEE Benelux Workshop on Model based Processing and Coding of Audio*, November 2002.

Bernhardsson, Erik. Implementing a Scalable Music Recommender System Ed. KTH CSC. 2009.

Bertin-mahieux, Thierry, François Maillet, Douglas Eck, and Paul Lamere. Autotagger: A Model For Predicting Social Tags from Acoustic Features on Large Music Databases. 2008. [cited 4 July 2010]. Available from world wide web: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.141.4606>.

Bialecki, Andrzej. Nutch, web-scale search engine toolkit. November 2009. Available from world wide web: <http://wiki.apache.org/nutch/Presentations?action=AttachFile&do=get&target=apachecon09.pdf>.

Cafarella, M., and D. Cutting. Building nutch: Open source search. *Queue* 2, 2004, 61.

Celma, O. Foafing the music: Bridging the semantic gap in music recommendation. *The Semantic Web-ISWC 2006* 2006, 927–934.

Celma, Oscar, Mohamed Sordo, Bramde Jong, XavierSerra, and Elena Martınez. Extending the folksonomies of freesound.org using content-based audio analysis. In *Proceedings of the Sound and Music Computing Conference*, [Porto, Portugal], 2009.

Decoste, Dennis et al. Recommender Systems Research at Yahoo! Research Labs. In *Proceedings of Beyond Personalization: The Next Stage of Recommender Systems*

Research, 2005.

Dittus, Martin. Hadoop at Last.fm. August 2008. Available from world wide web: <http://skillsmatter.com/custom/presentations/martin_dittus_hadoop_at_last.fm_overview.pdf>.

Jalote, P., A. Palit, and P. Kurien. The Timeboxing process model for iterative software development. *Advances in Computers* 62, 2004, 67–103.

Khare, R., D. Cutting, K. Sitaker, and A. Rifkin. Nutch: A flexible and scalable open-source web search engine. *Oregon State University* 2004.

Kreitz, G., and F. Niemela. Spotify–Large Scale, Low Latency, P2P Music-on-Demand Streaming. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, 1–10, 2010.

Lessig, Lawrence. *Free culture: the nature and future of creativity.* Penguin Books, 2004.

Linden, Greg, Brent Smith, and Jeremy York. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7, 2003, 76-80.

Luke Barrington; Reid Oda; Gert Lanckriet. SMARTER THAN GENIUS? HUMAN EVALUATION OF MUSIC RECOMMENDER SYSTEMS. In *In 10th International Society for Music Information Retrieval Conference*, 2009 Available from world wide web: <http://cosmal.ucsd.edu/cal/pubs/Barrington-Genius-ISMIR09.pdf>.

McNee, Sean M., John Riedl, and Joseph A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06 extended abstracts on Human factors in computing systems*, 1097-1101, [Montréal, Québec, Canada]: ACM, 2006 [cited 4 July 2010]. Available from world wide web: <http://portal.acm.org/citation.cfm?id=1125659>.

Michael, M., J. E Moreira, D. Shiloach, and R. W Wisniewski. Scale-up x scale-out: A case study using nutch/lucene. In *IEEE International Parallel and Distributed Processing Symposium, 2007. IPDPS 2007*, 1–8, 2007.

Middleton, Christian, and Ricardo Baeza-yates. A Comparison of Open Source Search Engines. [cited 16 September 2010]. Available from world wide web: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.119.6955>.

Moreira, J. E et al. Scalability of the Nutch search engine. In *Proceedings of the 21st annual international conference on Supercomputing*, 12, 2007.

Moreira, José E. et al. Scalability of the Nutch search engine. In *Proceedings of the 21st annual international conference on Supercomputing*, 3-12, [Seattle, Washington]: ACM, 2007 [cited 24 September 2010]. Available from world wide web:

<http://portal.acm.org/citation.cfm?id=1274975>.

Paroline, Jay. Technology Stack « Jay Paroline – Grooveshark Dev. May 2010. [cited 27 September 2010]. Available from world wide web: <http://wanderr.com/jay/technology-stack/2010/05/06/>.

Richard, Jones. Last.fm – the Blog · Last.fm Radio Announcement. *Last.fm Radio Announcement* March 2009. [cited 27 September 2010]. Available from world wide web: <http://blog.last.fm/2009/03/24/lastfm-radio-announcement>.

Shvachko, K., H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File system. *26th IEEE (MSST2010) Symposium on Massive Storage Systems and Technologies* 2010.

Singh, Vik. A Comparison of Open Source Search Engines « Vik Singh. July 2009. [cited 26 September 2010]. Available from world wide web: <http://zooie.wordpress.com/2009/07/06/a-comparison-of-open-source-search-engines-and-indexing-twitter/>.

11 Annex

11.1 Python web scrapping file clongclongmoo.py

```
import urllib2
from BeautifulSoup import BeautifulSoup

page = urllib2.urlopen("http://www.clongclongmoo.org/system/index.php?
cat=00_netaudio&page=05_netlabellist")
soup = BeautifulSoup(page)
content = soup.find('div', { "class" : "include" })

for tag in content.findAll('td', { "class" : "farbe-label" }):
    print tag.a['href']
```

11.2 Nutch configuration file nutch-site.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property> <name>generate.max.per.host</name>
<value>100</value>
</property>


<property>
  <name>http.agent.name</name>
  <value>Nutch crawler ADETTI</value>
  <description>HTTP 'User-Agent' request header.</description>
</property>

<property>
  <name>http.robots.agents</name>
  <value>Nutch crawler,*</value>
  <description>The agent strings we'll look for in robots.txt files,
  comma-separated, in decreasing order of precedence. You should
  put the value of http.agent.name as the first agent name, and keep the
  default * at the end of the list. E.g.: BlurflDev,Blurfl,*
  </description>
</property>


<property>
  <name>searcher.dir</name>
  <value>/home/andrericardo/apache-nutch-1.1/crawl-mp3</value>
  <description>
  Path to root of crawl.  This directory is searched (in
  order) for either the file search-servers.txt, containing a list of
  distributed search servers, or the directory "index" containing
  merged indexes, or the directory "segments" containing segment
  indexes.
  </description>
</property>

<property>
  <name>plugin.folders</name>
  <!-- <value>/home/andrericardo/apache-nutch-
1.1/src/plugin,plugins,build/plugins</value> -->
  <value>/home/andrericardo/apache-nutch-
1.1/src/plugin,plugins,build/plugins</value>
  <description>Directories where nutch plugins are located.  Each
  element may be a relative or absolute path.  If absolute, it is used
  as is.  If relative, it is searched for on the classpath.</description>
</property>


<property>
  <name>plugin.includes</name>
  <value>nutch-extensionpoints|protocol-http|urlfilter-regex|parse-(text|
html|js|tika)|index-(basic|anchor|more|mp3)|query-(basic|site|url|more|
mp3FieldQueryFilter)|response-(json|xml)|summary-basic|scoring-opic|
urlnormalizer-(pass|regex|basic)|creativecommons</value>
  <description>Regular expression naming plugin directory names to
  include.  Any plugin not matching this expression is excluded.
  In any case you need at least include the nutch-extensionpoints plugin.
By
  default Nutch includes crawling just HTML and plain text via HTTP,
  and basic indexing and search plugins. In order to use HTTPS please
enable
  protocol-httpclient, but be aware of possible intermittent problems with
the
  underlying commons-httpclient library. Nutch now also includes
integration with Tika
  to leverage Tika's parsing capabilities for multiple content types. The
existing Nutch
  parser implementations will likely be phased out in the next release or
```

```
so, as such, it is
  a good idea to begin migrating away from anything not provided by parse-
tika.
  </description>
</property>

<property>
  <name>query.basic.artist.boost</name>
  <value>2.22</value>
  <description> Declares a custom field and its boost to be added to the
default fields of the Lucene query.
  </description>
</property>

<property>
  <name>query.basic.album.boost</name>
  <value>0.33</value>
  <description> Declares a custom field and its boost to be added to the
default fields of the Lucene query.
  </description>
</property>

<property>
  <name>query.basic.genre.boost</name>
  <value>4.44</value>
  <description> Declares a custom field and its boost to be added to the
default fields of the Lucene query.
  </description>
</property>


 <property>
  <name>query.artist.boost</name>
  <value>0.0</value>
  <description> Used as a boost for cc field in Lucene query.
  </description>
</property>
<property>
  <name>query.title.boost</name>
  <value>1.25</value>
  <description> Used as a boost for title field in Lucene query.
  </description>
</property>

<property>
  <name>db.ignore.external.links</name>
  <value>true</value>
  <description>If true, outlinks leading from a page to external hosts
  will be ignored. This is an effective way to limit the crawl to include
  only initially injected hosts, without creating complex URLFilters.
  </description>
</property>

<property>
  <name>file.content.limit</name>
  <value>1048576</value>
  <description>The length limit for downloaded content, in bytes.
  If this value is nonnegative (>=0), content longer than it will be
truncated;
  otherwise, no truncation at all.
  </description>
</property>

</configuration>
```

11.3 Nutch plugin index-mp3

11.3.1 Plugin structure

```
index-mp3/
|-- build.xml
|-- plugin.xml
`-- src
    `-- java
        `-- org
            `-- apache
                `-- nutch
                    `-- indexer
                        `-- mp3
                            `-- Mp3IndexingFilter.java
```

11.3.2 Indexing filter Mp3IndexingFilter.java

```
package org.apache.nutch.indexer.mp3;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import org.apache.lucene.document.DateTools;

import org.apache.nutch.metadata.Nutch;
import org.apache.nutch.parse.Parse;

import org.apache.nutch.indexer.IndexingFilter;
import org.apache.nutch.indexer.IndexingException;
import org.apache.nutch.indexer.NutchDocument;

import org.apache.nutch.indexer.lucene.LuceneWriter;
import org.apache.hadoop.io.Text;

import org.apache.nutch.crawl.CrawlDatum;
import org.apache.nutch.crawl.Inlinks;

import java.net.MalformedURLException;
import java.net.URL;
import org.apache.hadoop.conf.Configuration;

import org.apache.nutch.metadata.Metadata;

public class Mp3IndexingFilter implements IndexingFilter {


    private static final Log LOG =
LogFactory.getLog(Mp3IndexingFilter.class);
    private static final String MP3_TRACK_TITLE = "track_title";
    private static final String MP3_ALBUM = "album";
    private static final String MP3_ARTIST = "artist";
    private static final String MP3_GENRE = "genre";
    private static final String MP3_RELEASE_DATE = "releaseDate";

    private Configuration conf;


    public NutchDocument filter(NutchDocument doc, Parse parse, Text url,
            CrawlDatum datum, Inlinks inlinks) throws IndexingException {

        // look up email of the author based on the url of the site
        //String creatorEmail =
EmailLookup.getCreatorEmail(url.toString());

      Metadata metadata = parse.getData().getParseMeta();

      String mp3Title = metadata.get("title");
      String mp3Album = metadata.get("xmpDM:album");
      String mp3Artist = metadata.get("xmpDM:artist");
      String mp3Genre = metadata.get("xmpDM:genre");
      String mp3releaseDate = metadata.get("xmpDM:releaseDate");

        LOG.info("######## mp3Title = " + mp3Title);
        LOG.info("######## mp3Album = " + mp3Album);
        LOG.info("######## mp3Artist = " + mp3Artist);
        LOG.info("######## mp3Genre = " + mp3Genre);
        LOG.info("######## mp3releaseDate = " + mp3releaseDate);

        if (mp3Title != null) {
            doc.add(MP3_TRACK_TITLE, mp3Title);
        }


        if (mp3Album != null) {
            doc.add(MP3_ALBUM, mp3Album);
        }

        if (mp3Artist != null) {
            doc.add(MP3_ARTIST, mp3Artist);
        }

        if (mp3Genre != null) {
            doc.add(MP3_GENRE, mp3Genre);
        }

        if (mp3releaseDate != null) {
            doc.add(MP3_RELEASE_DATE, mp3releaseDate);
        }

        return doc;
    }

    public void addIndexBackendOptions(Configuration conf) {

        LuceneWriter.addFieldOptions(MP3_TRACK_TITLE,
LuceneWriter.STORE.YES,
                LuceneWriter.INDEX.TOKENIZED, conf);

        LuceneWriter.addFieldOptions(MP3_ALBUM, LuceneWriter.STORE.YES,
                LuceneWriter.INDEX.TOKENIZED, conf);

        LuceneWriter.addFieldOptions(MP3_ARTIST, LuceneWriter.STORE.YES,
                LuceneWriter.INDEX.TOKENIZED, conf);

        LuceneWriter.addFieldOptions(MP3_GENRE, LuceneWriter.STORE.YES,
                LuceneWriter.INDEX.TOKENIZED, conf);

        LuceneWriter.addFieldOptions(MP3_RELEASE_DATE,
LuceneWriter.STORE.YES,
                LuceneWriter.INDEX.TOKENIZED, conf);

    }

    public Configuration getConf() {
        return conf;
    }

    public void setConf(Configuration conf) {
        this.conf = conf;
    }

}
```

11.3.3 build.xml

```
<?xml version="1.0"?>
<project name="index-mp3" default="jar-core">

  <import file="../build-plugin.xml"/>

</project>
```

11.3.4 plugin.xml

```
<plugin
   id="index-mp3"
   name="MP3 Indexing Filter"
   version="1.0.0"
   provider-name="andrericardo">

   <runtime>
      <library name="index-mp3.jar">
         <export name="*"/>
      </library>
   </runtime>

   <requires>
      <import plugin="nutch-extensionpoints"/>
   </requires>

   <extension id="org.apache.nutch.indexer.basic"
              name="Nutch MP3 Indexing Filter"
              point="org.apache.nutch.indexer.IndexingFilter">
      <implementation id="Mp3IndexingFilter"
                      class="org.apache.nutch.indexer.mp3.Mp3IndexingFilter
"/>
   </extension>

</plugin>
```