Architecture of The Portuguese Web Archive
Search System version 2

Fernando Melo, Hugo Viana, Daniel Gomes and Miguel Costa

August 30, 2016

Arquivo.pt
Abstract

Arquivo.pt - The Portuguese Web Archive, is a free and public service that enables full-text search over more than 2 billion files archived since 1996.

In September 2013, the entire system of Arquivo.pt collapsed and information was irreparably lost. Besides, Arquivo.pt also had to renovate its team.

This situation caused several problems on the reconstruction and maintenance of the search system. During the recovery process of the service, a severe lack of technical documentation about the architecture of the search system was identified.

This report describes the software architecture of the Arquivo.pt search system. The main difference from other web archiving systems is that it makes use of Lucene indexes to support full-text search and also URL search instead of the commonly used CDX files.

This report is divided in 8 chapters. First, it presents use cases that demonstrate how to use full-text, URL or advanced searches. Second, it describes the components and the interactions among them. Third it explains the software components and workflows. It follows a more detailed description about the Arquivo.pt software implementation. Chapter 5 presents the main operational procedures to maintain the system, such as how to deploy a collection and their indexes to production. Chapter 6 outlines which are the ranking features that are currently ordering the full-text search results. Chapter 7 presents the APIs of Arquivo.pt’s and finally chapter 8 describes the PyWb, component responsible for the replay of the archived Web pages.
Contents

Abstract 2

1 User Interfaces 10

2 Component-Based overview 15

2.1 System components 15

2.1.1 Search system of Arquivo.pt 15

2.1.1.1 Broker 15

2.1.1.2 QueryServer 16

2.1.1.3 DocumentServer 16

2.1.1.4 LVS: Load Balancer 16

2.1.2 Info about Arquivo.pt service 16

2.1.2.1 Info about the Arquivo.pt service 16

2.1.2.2 LVS: fail over 16

2.2 An overview of the Arquivo.pt workflow 16

2.2.1 Searching by URL or terms 17

2.2.2 Knowing more about the service 18

3 Software Components and the Workflows 19

3.1 Software Components 19

3.2 Workflows 20

4 Arquivo.pt Software Implementation 25

4.1 System Components 25

4.1.1 Broker 25

4.1.2 QueryServer 29
List of Figures

1.3 Full-text results page for the query "fccn" [http://arquivo.pt/search.jsp?l=pt&query=fccn.pt]. 12
1.4 Datepicker adapted to web archive users. 12
1.5 Advanced full-text search page [http://arquivo.pt/advanced.jsp]. 13
2.1 Workflow for searching by terms or URL. 17
2.2 Workflow for knowing more about the service. 18
3.1 Software architecture of Arquivo.pt. 21
4.1 The architecture of Arquivo.pt. 27
4.2 Workflow for a full-text search. 31
4.3 Workflow between Broker and QueryServer. 32
4.4 Workflow for a URL search. 33
4.5 Workflow for a page replay. 36
5.1 The two parts of an inverted index. The Dictionary is commonly kept in memory, with pointers to each posting list, which is stored on disk. (Retrieved from http://nlp.stanford.edu/IR-book/html/htmledition/an-example-information-retrieval-problem-1.html)

5.2 QueryServer flowchart for deploying a collection

5.3 Search results list when the indexes were properly set up.

5.4 DocumentServer flowchart for deploying a collection

5.5 Page replay of a DocumentServer

5.6 Result of a search-server.txt with a non coherent order

5.7 Removing access with Apache HTTP rules

5.8 Screenshot of first page of Luke

5.9 Opensearch header result for the query http://sapo.ua.pt


6.1 Webpage provided for testing ranking features

6.2 searchTests.jsp: Query-dependent features

6.3 searchTests.jsp: Query-independent features

6.4 searchTests.jsp: Temporal features

6.5 Boosting new features

8.1 config.yaml for pyWb (replay)

8.2 unframed.yaml for pyWb (noFrame)

8.3 µWsgi.ini for pyWb (replay)

8.4 µwsgi configurations in apache Httpd, located in /etc/httpd/vhosts.d/arquivo.conf

8.5 µwsgi connections with apache Httpd
List of Tables

3.1 An ArcProxy database populated .................................. 20
3.2 Protocols used by Arquivo.pt applications .......................... 23
3.3 URL’s to invoke directly each Tomcat web application hosted on Broker .................................................. 24
4.1 Tomcat logs generated by Broker ................................... 28
4.2 HTTP logster logs generated on the Broker .......................... 29
4.3 Rewrite rules to turn a web content inaccessible .................. 29
4.4 Logs generated on file named slave-searcher-10011.log .......... 30
4.5 Parameters broadcasted across QueryServer (full-text search) .......... 32
4.6 Parameters broadcasted across QueryServers (URL search) .... 35
4.7 Parameters broadcasted across QueryServers (advanced search) .... 37
5.1 Bundle of files necessary for deploying indexes (QueryServer) ...... 43
5.2 Example of QueryServer: search-servers.txt ....................... 43
5.4 Example of Broker: search-servers.txt ............................. 43
5.3 Bundle of files necessary for deploying indexes (Broker) ........ 44
5.5 Bash script to set up the ArcProxy ................................. 45
5.6 Excerpt from search-servers.txt on P58.arquivo.pt (Broker) ... 45
5.7 Excerpt from search-servers.txt on p55.arquivo.pt (QueryServer) .... 46
5.8 Excerpt from wrongly search-servers.txt (QueryServer) .......... 46
6.1 Ranking features in production in Arquivo.pt ....................... 65
### List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3</td>
<td>Lucene Similarity [6]</td>
<td>56</td>
</tr>
<tr>
<td>6.4</td>
<td>Lucene Similarity Normalized [7]</td>
<td>56</td>
</tr>
<tr>
<td>6.5</td>
<td>Nutchwax Similarity [8]</td>
<td>56</td>
</tr>
<tr>
<td>6.6</td>
<td>Nutchwax Similarity Normalized [9]</td>
<td>57</td>
</tr>
<tr>
<td>6.7</td>
<td>Computing Distances [10]</td>
<td>58</td>
</tr>
<tr>
<td>6.9</td>
<td>Url Depth [12]</td>
<td>61</td>
</tr>
<tr>
<td>6.11</td>
<td>LinInlinks [14]</td>
<td>61</td>
</tr>
<tr>
<td>6.13</td>
<td>Java code for SpanVersions feature [16]</td>
<td>63</td>
</tr>
<tr>
<td>6.14</td>
<td>NumberVersions [17]</td>
<td>63</td>
</tr>
<tr>
<td>6.15</td>
<td>BoostOlder [18]</td>
<td>63</td>
</tr>
<tr>
<td>6.16</td>
<td>BoostNewerAndOlder [19]</td>
<td>64</td>
</tr>
</tbody>
</table>
Chapter 1

User Interfaces

The Portuguese Web Archive project began in 2008 and it aims to preserve web contents of interest to the Portuguese community. It was based on the Internet Archive archive-access project tools [1], which are used by most web archives worldwide. However, we observed that these tools did not fulfill users requirements at several levels, from data integration to user interface design. Thus, we researched and developed a new web archive search engine [46].

This chapter illustrates the usage of Arquivo.pt through its several user interfaces. Arquivo.pt entry page is illustrated on fig. 1.1.

Figure 1.1: Home page of Arquivo.pt [http://arquivo.pt/index.jsp?l=en].

The entry page contains a highlights area which has examples of important web content archived, and a video that provides an usage example about the service Arquivo.pt. Provides a bilingual interface (i.e. Portuguese or English).

Arquivo.pt was designed to resemble a live-web search engine, in order to produce a familiar search experience to users that are new to web archiving.
services. A live-web search engine generally supports full-text or URL searches. *Arquivo.pt* provides a similar experience but adding historical features. Fig. 1.2 illustrates the historical search result page for the URL *fccn.pt*. The results page contains a table with all of the archived versions for the provided URL.

Figure 1.2: Url results page for the website "fccn.pt" ([http://arquivo.pt/search.jsp?l=pt&query=fccn.pt](http://arquivo.pt/search.jsp?l=pt&query=fccn.pt)).

![Arquivo.pt](image)

**Versions of the archived web pages**


<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Mar</td>
<td>31 Jan</td>
<td>31 Mar</td>
<td>31 May</td>
<td>31 Jun</td>
<td>1 Jan</td>
<td>1 Jan</td>
<td>12 Mar</td>
<td>23 Jun</td>
<td>31 Mar</td>
<td>20 Jan</td>
<td>20 Jan</td>
<td>20 Jan</td>
<td>20 Jan</td>
</tr>
<tr>
<td>2 Mar</td>
<td>2 Feb</td>
<td>3 Jun</td>
<td>10 Feb</td>
<td>15 Apr</td>
<td>7 Jan</td>
<td>6 Jan</td>
<td>11 Jan</td>
<td>12 Mar</td>
<td>23 Jun</td>
<td>31 Mar</td>
<td>20 Jan</td>
<td>20 Jan</td>
<td>20 Jan</td>
</tr>
<tr>
<td>28 May</td>
<td>1 May</td>
<td>27 Sep</td>
<td>9 Aug</td>
<td>6 Jun</td>
<td>20 Jan</td>
<td>19 Jan</td>
<td>26 Jan</td>
<td>1 Oct</td>
<td>4 Aug</td>
<td>4 Aug</td>
<td>12 Apr</td>
<td>12 Apr</td>
<td>12 Apr</td>
</tr>
<tr>
<td>7 Jun</td>
<td>7 Mar</td>
<td>23 Sep</td>
<td>10 Oct</td>
<td>11 Jun</td>
<td>12 Jan</td>
<td>17 Jan</td>
<td>27 Jan</td>
<td>12 Dec</td>
<td>4 Aug</td>
<td>12 May</td>
<td>12 May</td>
<td>12 May</td>
<td>12 May</td>
</tr>
<tr>
<td>7 Jul</td>
<td>9 Apr</td>
<td>20 Nov</td>
<td>24 Nov</td>
<td>13 Jun</td>
<td>6 Feb</td>
<td>9 Feb</td>
<td>9 Feb</td>
<td>10 Dec</td>
<td>21 May</td>
<td>21 May</td>
<td>21 May</td>
<td>21 May</td>
<td>21 May</td>
</tr>
<tr>
<td>16 Aug</td>
<td>17 Aug</td>
<td>18 Jun</td>
<td>9 Feb</td>
<td>7 Feb</td>
<td>7 Feb</td>
<td>7 Feb</td>
<td>18 Dec</td>
<td>21 May</td>
<td>21 May</td>
<td>21 May</td>
<td>21 May</td>
<td>21 May</td>
<td>21 May</td>
</tr>
</tbody>
</table>

A user can navigate across the web archived content and analyze its evolution through the years, as a time machine of web content.

Fig. 1.3 illustrates the full-text results page for the term "fccn". The results list contains archived web content with the word "fccn".

Two graphical datepickers are provided (both in URL and full-text queries) in order to delimit the timespan of a given query. These Datepickers are based on the Datepicker of jQuery UI library, but adapted to users of web archiving services [45].

Fig. 1.4 illustrates the graphical datepicker developed for the web archive paradigm.

Fig. 1.5 presents the advanced search page where users can apply several
Figure 1.3: Full-text results page for the query ‘fccn’ (http://arquivo.pt/search.jsp?l=en&query=fccn).

Figure 1.4: Datepicker adapted to web archive users.

It contains the following fields:

**Words**

*With these words:* Looks up for web content which contains these words;

*With these phrase:* Looks up for web content which contains this phrase;

*Without any of these phrase:* Looks up for web content which does not contain this phrase.

**Date**

*Between:* Looks up for web content by delimiting the date;
Sort by: Sets up the sort order.

Format

Show the pages in the format: Looks up for web content which are on a specific format (i.e. HTML, PDF etc..);

Website

With this address: Looks up for web content which contains this address;

Number of results

show: Sets up the number of results per page to be shown.

Figure 1.5: Advanced full-text search page [http://arquivo.pt/advanced.jsp].

Fig. 1.6 illustrates the informative home page of *Arquivo.pt*. It contains, for instance, publications related with the *Arquivo.pt*.

Figure 1.6: Informative site (http://sobre.arquivo.pt/portuguese-web-archive-2?set_language=en).
Chapter 2

Component-Based overview

2.1 System components

This section introduces a slight overview about the components of Arquivo.pt. It begins with an explanation of each component and then the workflow among them.

Arquivo.pt is divided in two platforms, which are:

Search system: is the full-text and URL search content; (http://arquivo.pt/?l=en)

Info about the service: is the Arquivo.pt Content Management System (CMS). (http://sobre.arquivo.pt/?set_language=en)

2.1.1 Search system of Arquivo.pt

This subsection describes the components of the Arquivo.pt search system.

2.1.1.1 Broker

The Broker is responsible for delivering the ranked search results to users. The service contains redundancy between the Brokers, which means that two Brokers are listening and processing users requests. It also redirects the searches to the Query Servers, merges their results and selects the top ones.
2.1.1.2 QueryServer

QueryServer is the component that stores, looks up the indexes, ranks the results and then sends the results to the Broker. The indexes are distributed across multiple QueryServers.

2.1.1.3 DocumentServer

DocumentServer is the component that stores the harvest of web content in ARC format and makes them accessible to the Broker. The harvest of web content are distributed across multiple DocumentServers.

2.1.1.4 LVS: Load Balancer

The load balancer balances the traffic distribution between the two Brokers. The service contains redundancy between the load balancer, which means two load balancers for processing users requests.

2.1.2 Info about Arquivo.pt service

This subsection describes the CMS component which manages all information about the service.

2.1.2.1 Info about the Arquivo.pt service

Arquivo.pt CMS contains several informations about the service, such as documents produced, papers submitted and news.

2.1.2.2 LVS: fail over

The fail over is a backup of the Arquivo.pt CMS in which the functions of the component are assumed by secondary system components when the primary component becomes unavailable.

2.2 An overview of the Arquivo.pt workflow

Based on the two platforms introduced in section 2.1, this section describes the workflow provided to users.
2.2.1 Searching by URL or terms

Figure 2.1: Workflow for searching by terms or URL.

1. The user submits a query to http://arquivo.pt/?l=en
2. The load balancer picks one of the two Brokers to process the requests;
3. The Broker processes the query submitted and then broadcast it across QueryServers;
4. The QueryServer looks up for the query on the search data structures;
5. The QueryServer sends the ranked results list to the Broker;
6. The Broker processes the request and locates which DocumentServer stores the archived web content. Then, it requests the archived web content to the DocumentServer;
7. The DocumentServer obtains the archived web content;
8. The DocumentServer streams the archived web content to Broker;
9. The Broker reproduces the archived web content to user’s browser.
2.2.2 Knowing more about the service

Figure 2.2: Workflow for knowing more about the service.


2. The failover checks if the primary components is available, if not the request is processed by the secondary;

3. The web site is sent to the user browser.
Chapter 3

Software Components and the Workflows

This chapter describes the bundle of software needed for setting up Arquivo.pt search system.

3.1 Software Components

The following describes the software components of Arquivo.pt.

nutchwax: manages the workflow of a URL search, processes full-text search requests and hosts the users interfaces;

PyWb (replay): responsible for the replay of the archived Web pages; manages the streaming web content sent from arcproxy, and processes a URL search request [48];

PyWb (noFrame): allows the replay of archived Web pages. This particular running instance, without the evolving frame, is used by the screenshot-Server and printServer.

screenshotServer: A PhantomJS Web Server that takes a screenshot of a given URL and returns a JSON with the base64 of the screenshot that was taken. This Web Server can be called when a user asks to print a Web page, followed by printing the screenshot image.
**printServer:** A PhantomJS Web Server that takes a screenshot of a given URL and returns a .png image to the browser.

**arcproxy:** hosts a Berkeley database that maps the arcfile name into the server location (table 3.1 contains an example) and provides a stream connection for supplying the web content to the pyWb waybacks;

**spellchecker:** is a mechanism that suggests alternatives queries (43);

**browser** provides navigation over archived web contents;

**nutchwax-job-0.11.0-SNAPSHOT.jar:** is a java standalone application for listening query requests from Broker and then returns a list with the 10k most relevant results. It also provides the hadoop jobs for the indexing process;

**Plone:** *Arquivo.pt* content management system;

**Zope:** Web application server for running Plone CMS;

**Tomcat:** Web application server for running java web applications;

**µWsgi:** Web application server for running python Web applications; runs the pyWb waybacks.

**Piranha:** Load-balanced generic service clustering environment;

<table>
<thead>
<tr>
<th>Name</th>
<th>Server path</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCCN-PT-HISTORICAL-ia400129.arc.gz</td>
<td>arquivo.pt:8080/browser/files/IAall/arcs/FCCN-PT-HISTORICAL-ia400129.arc.gz</td>
</tr>
<tr>
<td>FCCN-PT-HISTORICAL-ia400129.arc.gz</td>
<td>arquivo.pt:8080/browser/files/IAall/arcs/FCCN-PT-HISTORICAL-ia400129.arc.gz</td>
</tr>
<tr>
<td>FCCN-PT-HISTORICAL-ia400129.arc.gz</td>
<td>arquivo.pt:8080/browser/files/IAall/arcs/FCCN-PT-HISTORICAL-ia400129.arc.gz</td>
</tr>
</tbody>
</table>

**3.2 Workflows**

Fig. 3.1 details the software and the workflow of the *Arquivo.pt* service. The workflow of the software architecture is divided in:

**Terms Query:** Search by terms.
Figure 3.1: Software architecture of *Arquivo.pt*.

1. A user submits a terms query to Apache, which redirects it to nutchwax inside Tomcat.
2. The nutchwax asks the spellchecker for a query suggestion.
3. The spellchecker looks up the best suggestion and sends it to the nutchwax.
4. The nutchwax broadcasts the request, which contains the terms query, across the nutchwax-job-0.11.0-SNAPSHOT.jar cluster.
5. Each nutchwax-job-0.11.0-SNAPSHOT.jar accesses the index and responds with a 10k ranked list of documents.

**URL Query:** Search by URL.

1. The user submits the URL query to Apache, which redirects it to pyWB (replay) wayback inside µWsgi.
2. The nutchwax asks the spellchecker for a query suggestion.
3. The spellchecker looks up the best suggestion and sends it to the nutchwax.
4. The nutchwax redirects the request to the respective nutchwax-job-0.11.0-SNAPSHOT.jar.
5. The nutchwax-job-0.11.0-SNAPSHOT.jar accesses the index and responds with a list of URL versions.
6. The nutchwax redirects the list of URL versions to the pyWb (replay).

**Page View:** Web content reply.

1. The user submit the URL of the page to view to Apache. Apache redirects it to pyWB (replay) wayback inside µWsgi.
2. The pyWB (replay) redirects the request to nutchwax.
3. The nutchwax redirects the request to the nutchwax-job-0.11.0-SNAPSHOT.jar.
4. The nutchwax-job-0.11.0-SNAPSHOT.jar accesses the index and responds with an ARC filename and offset.
5. The pyWb wayback requests an ARC file to arcproxy.
6. The arcproxy accesses BerkeleyDB to get the URL of the ARC file.
7. The arcproxy gets the URL, which is a HTTP request to browser.
8. The browser returns the ARC file stored in the file system and then arcproxy starts streaming the web content to wayback. The wayback, in turns, stops the streaming when it gets the all of the requested web content.

**Screenshot Request:** Downloads a .png screenshot of a given Web page

1. A user submits a screenshot request of a given Web page. The request is made to the phantomjs screenshotServer.
2. The phantomjsServer loads an unframed version of the requested Web page in order to take a screenshot.
3. The pyWB (replay) redirects the request to nutchwax.
4. The nutchwax redirects the request to the nutchwax-job-0.11.0-SNAPSHOT.jar.
5. The nutchwax-job-0.11.0-SNAPSHOT.jar accesses the index and responds with an ARC filename and offset.
6. The pyWb unFramed requests an ARC file to arcproxy.
7. The arcproxy accesses BerkeleyDB to get the URL of the ARC file.
8. The arcproxy gets the URL, which is a HTTP request to browser.
9. The browser returns the ARC file stored in the file system and then arcproxy starts streaming the web content to wayback. The wayback, in turns, stops the streaming when it gets the all of the requested web content.
**Print Request:** Calls the default printer, printing a screenshot of a given Web page

1. A user submits a print request for a given Web page. The request is made to the phantomjs printServer.

2. The phantomjsServer loads an unframed version of the requested Web page in order to take a screenshot of the web page to be printed.

3. The pyWB (replay) redirects the request to nutchwax.

4. The nutchwax redirects the request to the nutchwax-job-0.11.0-SNAPSHOT.jar.

5. The nutchwax-job-0.11.0-SNAPSHOT.jar accesses the index and responds with an ARC filename and offset.

6. The pyWb unFramed requests an ARC file to arcproxy.

7. The arcproxy accesses BerkeleyDB to get the URL of the ARC file.

8. The arcproxy gets the URL, which is a HTTP request to browser.

9. The browser returns the ARC file stored in the file system and then arcproxy starts streaming the web content to wayback. The wayback, in turns, stops the streaming when it gets the all of the requested we

Table 3.2 presents, for each web application, which are the communication protocols used for exchanging messages.

<table>
<thead>
<tr>
<th>Application name</th>
<th>pyWb</th>
<th>nutchwax</th>
<th>arcproxy</th>
<th>hadoopRpcServer</th>
<th>browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>pyWb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nutchwax</td>
<td>HTTP:8080</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arcproxy</td>
<td>HTTP:8080</td>
<td>HadoopRPC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hadoopRpcServer</td>
<td></td>
<td>HadoopRPC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>browser</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HTTP:8080</td>
</tr>
</tbody>
</table>

Table 3.3 outlines how to use each Arquivo.pt servlet. For instance, to call the webapp wayback (pyWb replay) we have to produce the following:

`wayback/19961013211156/http://www.telepac.pt/filosoft/teste.zip`
Table 3.3: URL’s to invoke directly each Tomcat web application hosted on Broker.

<table>
<thead>
<tr>
<th>Webapp name</th>
<th>URL example</th>
</tr>
</thead>
<tbody>
<tr>
<td>wayback/</td>
<td>19961013211156/<a href="http://www.telepac.pt/filosoft/teste.zip">http://www.telepac.pt/filosoft/teste.zip</a> (pyWb replay)</td>
</tr>
<tr>
<td>noFrame/replay/</td>
<td>19961013211156/<a href="http://www.telepac.pt/">http://www.telepac.pt/</a> (pyWb noFrame)</td>
</tr>
<tr>
<td>arcproxy/</td>
<td>AWP-Itotero-2005051028156-00000.arc.gz</td>
</tr>
<tr>
<td>spellchecker/</td>
<td>checker?query=hola</td>
</tr>
<tr>
<td>browser/</td>
<td>files/FAWP10/FAWP-01-07-12-20120701150129883/arcs/IAH-20120701150204-00027-p12.arquivo.pt.arc.gz</td>
</tr>
</tbody>
</table>
Chapter 4

Arquivo.pt Software Implementation

Fig. 4.1 details the components architecture of the Arquivo.pt.

4.1 System Components

This section details the software components of Arquivo.pt.

4.1.1 Broker

The Broker is "the manager" of the Arquivo.pt. The following web applications are hosted on the Broker:

nutchwax
pywb (replay)
pywb (noFrame)
printServer
screenshotServer
arcproxy
spellchecker
For further information about the web application, see chapter 3 that explains every web application used in Arquivo.pt.
Figure 4.1: The architecture of Arquivo.pt.
The Broker contains the following log files:

**ArcProxy:**BerkeleyDB  (more info in chapter 3)

**catalina.out:**Tomcat logs about query submission or components workflows are registered. For changing the log4j levels the following files are needed:

- **wayback** /wayback/WEB-INF/classes/log4j.properties
- **nutchwax** /nutchwax/WEB-INF/classes/log4j.properties
- **arcproxy** /arcproxy/WEB-INF/classes/log4j.properties
- **spellchecker** /spellchecker/WEB-INF/classes/log4j.properties

Table 4.1 contains examples of generated logs on catalina.out.

Table 4.1: Tomcat logs generated by Broker.

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Code</th>
<th>Method</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feb 05, 2016</td>
<td>1:14:29PM</td>
<td>INFO</td>
<td>logNotInArchive</td>
<td><a href="http://gold.org/investment/statistics/gold_price_chart">http://gold.org/investment/statistics/gold_price_chart</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Retrying connect to server:</td>
<td>Already tried 1 time(s).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>/home/wayback/searcher/apache-tomcat-5.5.25/webapps/nutchwax/WEB-INF/classes/plugins</td>
<td></td>
</tr>
</tbody>
</table>

**http_logster:**HTTPD logs of HTTP request to Arquivo.pt. Table 1.2 presents several entries of http_logster file.
Table 4.2: HTTP_logster logs generated on the Broker.

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Request URL</th>
<th>Status Code</th>
<th>Bytes Retired</th>
<th>Bytes Requested</th>
<th>Time Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>'GET /spellchecker/checker?query=teste&amp;l=pt HTTP/1.1'</td>
<td>500</td>
<td>865</td>
<td>4384</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET / HTTP/1.1'</td>
<td>200</td>
<td>3275</td>
<td>347499</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET /img/highlights/figo.png HTTP/1.1'</td>
<td>200</td>
<td>1012</td>
<td>1936</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET /img/search-icon.gif HTTP/1.1'</td>
<td>200</td>
<td>2922</td>
<td>8891</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET /img/mec-web.png HTTP/1.1'</td>
<td>200</td>
<td>3309</td>
<td>1775</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET /img/logo-pos.gif HTTP/1.1'</td>
<td>200</td>
<td>3077</td>
<td>1799</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET /img/logo-fccn.png HTTP/1.1'</td>
<td>200</td>
<td>2218</td>
<td>6274</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET /img/language-arrow.gif HTTP/1.1'</td>
<td>200</td>
<td>114</td>
<td>1696</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET /img/search-reset.gif HTTP/1.1'</td>
<td>200</td>
<td>268</td>
<td>1996</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET /img/search-inputtext.png HTTP/1.1'</td>
<td>200</td>
<td>211</td>
<td>2030</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET /img/search-submit.gif HTTP/1.1'</td>
<td>200</td>
<td>1619</td>
<td>1776</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET /img/box-background.gif HTTP/1.1'</td>
<td>200</td>
<td>4587</td>
<td>1802</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET /img/box-mask.png HTTP/1.1'</td>
<td>200</td>
<td>358</td>
<td>2104</td>
<td></td>
</tr>
<tr>
<td>127.0.0.1</td>
<td>'GET /search.jsp?l=pt&amp;query=porcaria&amp;btnSubmit=Pesquisar+no+Arquivo HTTP/1.1'</td>
<td>200</td>
<td>5444</td>
<td>2766328</td>
<td></td>
</tr>
</tbody>
</table>

Blacklist_DocServer:HTTPD_mod_rewrite configuration file to manage accesses constraints of web content (apache HTTPD rewrite rules).

Table 4.3 presents several rewrites rules.

Table 4.3: Rewrite rules to turn a web content inaccessible.

RewriteRule /wayback/id30257363index0 - [R=404,NC,L]
RewriteRule /wayback/id105349813index0 - [R=404,NC,L]
RewriteRule /wayback/id105560458index0 - [R=404,NC,L]

4.1.2 QueryServer

QueryServer is the component that stores and manages the indexes. The collection’s indexes are distributed across multiple QueryServers.

QueryServer hosts the following java application.

nutchwax-job-0.11.0-SNAPSHOT.jar (more info in chapter 3)

The QueryServer contains the following log files:
Index and Segments: stores the indexes and segments.

SERP_Blacklist: list of removed webpages from Arquivo.pt;

slave-searcher-$Collection$Port:HadoopRPCServer logs about users queries.

Table 4.4 illustrates a few entries of generated logs on slave-searcher-$Collection$Port.

Table 4.4: Logs generated on file named slave-searcher-10011.log.

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Level</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>15/12/16</td>
<td>10:42:24</td>
<td>INFO</td>
<td>ipc.Server: IPC Server handler 19 on 10016: starting</td>
</tr>
<tr>
<td>15/12/16</td>
<td>10:42:24</td>
<td>INFO</td>
<td>ipc.Server: IPC Server handler 18 on 10016: starting</td>
</tr>
<tr>
<td>15/12/16</td>
<td>10:45:01</td>
<td>INFO</td>
<td>searcher.LuceneQueryOptimizer: +date:[0820454400 TO 1420070399]ˆ0.0</td>
</tr>
<tr>
<td>16/02/05</td>
<td>16:07:10</td>
<td>INFO</td>
<td>searcher.LuceneQueryOptimizer: +exacturl:T4LL2V34WO5DK5HL2VHCAGHLK4ˆ0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+exacturl:LH5XGGMZBI2FO7OMTJ7BQE3UQA0.1</td>
</tr>
</tbody>
</table>

4.1.3 DocumentServer

The DocumentServer is the component that stores the harvest of web content crawled in ARC format.

The following web applications are hosted on the DocumentServer:

browser (more info in chapter 3)

It contains the following log files:

ArcFiles:arc.gz stores the archived web content.

catalina.out:Tomcat logs about the accesses to the archived web content.

4.1.4 Arquivo.pt_CMS

Arquivo.pt_CMS is implemented using version 3.0.5 of Plone. Plone is a free and open source content management system (CMS) built on top of the Zope application server 20.
4.2 Workflow

This section provides details about the communication among QueryServers, Brokers and DocumentServers, through use cases. Arquivo.pt provides the following workflows for users:

1. sends a query submission by URL.
2. sends a query submission by terms.
3. sends a query submission through advanced search.
4. requests a web content view.

4.2.1 Full-text query

Fig. 4.2 illustrates the workflow when a full-text search query is submitted to Arquivo.pt.

Figure 4.2: Workflow for a full-text search.

\[\text{IN: 'query: Luis Figo'}\]

\[\text{OUT: List of documents which contains 'Luis Figo'}\]
1. User submits a query request to Apache HTTP server.
   `<http://arquivo.pt/search.jsp?query=%22Luis%20Figo%22>`

2. Inside the Broker: Apache HTTP server redirects it to nutchwax.
   `<http://arquivo.pt/nutchwax/opensearch?query="Luis%20Figo">`

3. Inside the Broker: The nutchwax broadcasts the processed request across the QueryServers. Fig. 4.3 contains the protocol fields broadcasted.

   **Figure 4.3: Workflow between Broker and QueryServer**

   The communication is done over the Hadoop RPC protocol. The request parameters are described in the table 4.5.

   **Table 4.5: Parameters broadcasted across QueryServer (full-text search).**

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>Luis(%20Figo</td>
<td>full-text search encoded with timestamp by default.</td>
</tr>
<tr>
<td>query</td>
<td>date:'19960101000000-20141231235959'</td>
<td></td>
</tr>
<tr>
<td>numHits</td>
<td>20</td>
<td>sets the number of listed results.</td>
</tr>
<tr>
<td>searcherMaxHits</td>
<td>10000</td>
<td>sets the max number of results for the posting list.</td>
</tr>
<tr>
<td>maxHitsPerDup</td>
<td>2</td>
<td>sets the max number of repeated results on the posting list.</td>
</tr>
<tr>
<td>dedupField</td>
<td>site</td>
<td>sets the maxHitsPerDup field filter.</td>
</tr>
<tr>
<td>sortField</td>
<td>null</td>
<td>sets the field name to sort (i.e sort by date ..).</td>
</tr>
<tr>
<td>reverse</td>
<td>false</td>
<td>sets the sort order ascending or descending.</td>
</tr>
<tr>
<td>rankingfunctions</td>
<td>49=0.5939482, 37=0.34503713, 45=1.2592824, 34=0.023322608</td>
<td>sets the ranking features and its weights.</td>
</tr>
<tr>
<td>maxHitsPerVersion</td>
<td>1</td>
<td>selects max number of repeated versions.</td>
</tr>
</tbody>
</table>

4. Each QueryServer: Returns a response that contains a posting list with the 10k most relevant results. (see figure 4.3)

5. Inside Broker: The nutchwax accesses the responses from the QueryServers and performs the list of documents in XML format. The search results list is returned to the user.
4.2.2 URL search

Fig. 4.4 illustrates the workflow when a URL search query is submitted to Arquivo.pt.

Figure 4.4: Workflow for a URL search.

Websites on the internet do not host the same entry page name and users do not know every entry page. Thereby, exacturl was developed to improve user’s experience and it expands an URL with the most common entry pages.

Arquivo.pt provides two ways of expanding a URL:

- exacturl
- exacturlexpand

The exacturlexpand was added by Arquivo.pt team, in order to expand more types of entry pages. For instance, an entry page named index.php or index.asp is expanded with exacturlexpand and it is not with exacturl. (2)

IN: ‘query: www.fccn.pt’

OUT: List of archived versions form "www.fccn.pt"
1. User submits a query to Apache HTTP server.

```
```

2. Inside Broker: Apache HTTP server redirects the request to nutchwax.

3. Inside Broker: nutchwax processes the request and redirects it to wayback with the url encoded.

```
```

4. Inside Broker: The wayback processes and redirects the request to nutchwax.

5. Inside Broker: nutchwax encodes the URL using the MessageDigest algorithm with Base32. The table 4.6 details the request parameters broadcasted to the QueryServers. (This custom encoding is necessary because NutchAnalysis will not let through ‘?’ or ‘=' characters in clause values and the ‘&’ character can’t be passed in a query string because it’ll confuse request.getParameter.)

6. Inside Broker: The nutchwax accesses the responses from QueryServers and performs the list of URL versions in XML format.

7. Inside Broker: The nutchwax redirects the list of URL version to wayback.

### 4.2.3 Page replay

Fig. 4.2 illustrates the workflow when a page replay is requested to Arquivo.pt.

**IN:** http://arquivo.pt/wayback/20130330011529/http://www.fccn.pt/

**IN:** Arcoffset:39099130

**IN:** Arcname: FCCN-PT-HISTORICAL-ia400125.20090108013027.arc.gz

**OUT:** Archived content

1. User submits the URL to Apache HTTP server.

```
```

2. Inside the Broker: Apache HTTP redirects the request to wayback.
Table 4.6: Parameters broadcasted across QueryServers (URL search).

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>'date:19960101000000-20141231235959'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td>exacturlexpand</td>
<td>'7S2KTVSC29FW0QKE3ALEFTW2O4'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'WU4341YSN6QJXXSK5ONOXPFI'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'4R4HDODIKUXEJKCA0SXFETF9S'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'HYRDHCA0HGC2BHXGIEFTY4QMI'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'ETW3VFLMD09H3Y0T7KQIP7T74'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'EB7TK4WMP2ASBIC3HF2SCHU2A'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'ZONLPBFHM2WTWF2EFGX36DZP3'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'6H575MUCMD6GFR6C2FZLWLM'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'IVK55INJAEURZRA2A4E2RMZJ4'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'NQJ7P3ZJXPM34YUXUD5G4'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'%WQ65BF2Z2QVMEMD3JUI7T7UY'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'DWDN4Q5MTZX3ETWQZ3Y8C3M1'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'P4VQPSJH6UEJMPDNC4I56WU1'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'WE54HNB55FOPHPXHAMDUEC2T4U'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'FZLAMUVMZIFGRQVNYAEMZM6HEX'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'DTHXCWKH7ZUCWQNA2D2PN34'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'ZFTVHM4404ULVWAJPGG2NPVB4'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'WBSMFHC3XZTYAP3MKTVESTT7E'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'FLA7WJ37F6U6E6QXZAWJ8H'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'NYMEERSLreddYaal2WEKJC3J51A76V5E'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'UI4CWF667X5IAZ5MTWQ2F4'</td>
<td>sets encoded query string</td>
</tr>
<tr>
<td></td>
<td>'4NDT7NAA4YZLEPS3YQEUKEWCAU'</td>
<td>sets encoded query string</td>
</tr>
</tbody>
</table>

| numHits        | 20                                                                      | sets the number of listed results.                                           |
| maxHitsPerDup  | 10000                                                                   | sets the max number of results for the posting list.                         |
| dermapField    | site                                                                    | sets the maxHitsPerDup field filter.                                         |
| sortField      | null                                                                    | sets the field name to sort (i.e sort by date ..).                           |
| reverse        | false                                                                   | sets the sort order by ascending or descending.                              |
| rankingfunctions | 49=0.5939482, 37=0.34503713, 45=1.2592824, 34=0.023322608                 | sets the ranking features and its weights.                                   |
| maxHitsPerVersion | 1                                                                     | selects max number of repeated versions.                                     |

3. Inside the Broker: The wayback redirects the request to nutchwax. The request contains the exacturlexpand and the timestamp.

4. Broker communication with QueryServer: The nutchwax broadcasts the request with exacturlexpand and timestamp across QueryServers.

5. Inside the Broker: The nutchwax processes the response from QueryServer and responds with the ARC filename and the offset to the wayback.

6. Inside the Broker: The wayback requests the web content using HTTP Range Requests [53] to arcproxy. The request contains the arcfile name and the arcoffset.

   <http://arquivo.pt/arcproxy/arcproxy/FCCN-PT-HISTORICAL-ia400125.20090108013027.arc.gz>,
   "Content-Range: bytes 39093130 - ">

7. Inside the Broker: The arcproxy accesses BerkeleyDB and gets the host-
name of the DocumentServer, that stores the arcfile.

8. Inside the Broker: The arcproxy requests the web content to the browser and then streams it to wayback.

9. Inside the Broker: The wayback stops the communication when the bytes streamed from arcproxy reached the content-length of the requested web content.

4.2.4 Advanced search

The advanced search is similar with full-text search, though with a nuance that you can perform your filter as you need. Hence, the fig. 4.2 also illustrates the workflow for an advanced search.

IN: 'With these words: sapo' Between: 20070101000000-20141231235959
Show the pages in the format: pdf With this address: www.sapo.pt
OUT: List of documents containing the filter choices

1. User submits the advanced query to Apache HTTP server which contains the performed fields.

2. On Broker: Apache HTTP redirects the request to nutchwax.

3. On Broker: nutchwax broadcasts the requests across QueryServers. Table 4.7 details the request parameters.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>sapo site:’www.sapo.pt’ type:’pdf’ date:’20070101000000-20141231235959’</td>
<td>set the query string with the performed filter.</td>
</tr>
<tr>
<td>numHits</td>
<td>20</td>
<td>sets the number of listed results.</td>
</tr>
<tr>
<td>searchMaxHits</td>
<td>10000</td>
<td>sets the max number of results for the posting list.</td>
</tr>
<tr>
<td>maxHitsPerDup</td>
<td>2</td>
<td>sets the max number of repeat results on the posting list.</td>
</tr>
<tr>
<td>dedupField</td>
<td>site</td>
<td>sets the maxHitsPerDup field filter.</td>
</tr>
<tr>
<td>sortField</td>
<td>null</td>
<td>sets the field name to sort (i.e sort by date ..).</td>
</tr>
<tr>
<td>reverse</td>
<td>false</td>
<td>sets the sort order by ascending or descending.</td>
</tr>
<tr>
<td>rankingfunctions</td>
<td>49=0.5939482, 37=0.34503713, 45=1.2592824, 34=0.023322608</td>
<td>sets the ranking features and its weights.</td>
</tr>
<tr>
<td>maxHitsPerVersion</td>
<td>1</td>
<td>selects max number of repeated versions.</td>
</tr>
</tbody>
</table>

4. Each QueryServer: Returns back a response that contains a posting list with the 10k most relevant results.

5. Inside Broker: The nutchwax responds a ranked list of documents.

4.2.5 URL syntax to reference a web content

Arquivo.pt provides two manners for accessing an URL. Lucene DocId syntax (no longer available) and the standarized wayback syntax, the follow are examples of both respectively:

DocId http://arquivo.pt/wayback/id24689index25


Lucene DocId syntax, the number 24689 is the DocId and the number 25 is the collection entry on search-servers.txt (i.e. for instance FAWP11). Wayback
syntax 19980205082901 is the timestamp for the web content. The format of the timestamp is 1-14 digits (YYYYMMDDhhmmss).
Chapter 5

Maintenance Procedures

This chapter describes procedures for maintaining Arquivo.pt search system system. It is divided in the following sections:

• Indexing a collection;
• Installing the Arquivo.pt search system;
• Deploying a collection;
• Removing archived web content from the search results;
• Looking up the Lucene-indexes with Luke;
• Installing the informative site (Arquivo.pt_CMS);

5.1 Indexing a collection

Arquivo.pt produces three types of collections, which are broad crawls (AWP), daily crawls (FAWP) and extraordinary crawls (EAWP).

AWP crawl done every 4 months;

FAWP crawl done every day;

EAWP crawl done in special occasions.

The software architecture of Arquivo.pt includes a Hadoop cluster (19 Computers) which is a computational cluster designed specifically for storing and
analyzing huge amounts of unstructured data in a distributed computing environment. The Hadoop cluster is used to index the harvest of web content and instructions for compiling and installing are available in (21) and (22). Arquivo.pt developed a bash script to execute all jobs named index-collection.sh. The steps of the script are explained in the subsection 5.1.2.

5.1.1 Indexing steps

Arquivo.pt contains a Hadoop job for the process of indexation (23).

The Hadoop job contains the following sub-jobs:

fawp_prepare: Creates an auxiliary DB with all of the gathered URLs. It is used when there are URLs with the same domain name but with different timespans (only for FAWP crawl).

import: Ingests ARCs writing ARC Record parse as Nutch FetcherOutputFormat.

update: Update dbs with recent imports. This process creates the crawldb directory on the HDFS. The crawldb maintains information on all known URLs (fetch schedule, fetch status or metadata).

invert: Invert links (create structure outlink->inlinks). This process creates the linkdb directory on the HDFS. The linkdb maintains an inverted link map, listing incoming links for each url.

index: This process creates the indexes directory on the HDFS.

merge: Merges indexes partitions into one. This process creates the index directory on the HDFS.

index_prunning: The posting lists of the five fields (content, title, anchor, url and host) are pruned.

Arquivo.pt removes postings that are not shown in results, in order to produce a much smaller index that returns identical results. Each item in the list - which records that a term appeared in a document - is conventionally called a posting. The list is called posting list. Fig. 5.1 illustrates an example of a posting and posting list.
Figure 5.1: The two parts of an inverted index. The Dictionary is commonly kept in memory, with pointers to each posting list, which is stored on disk. (Retrieved from http://nlp.stanford.edu/IR-book/html/htmledition/an-example-information-retrieval-problem-1.html)

| Brutus | → | 1 | 2 | 4 | 11 | 31 | 45 | 173 | 174 |
| Caesar | → | 1 | 2 | 4 | 5 | 6 | 16 | 57 | 132 | ... |
| Calpurnia | → | 2 | 31 | 54 | 101 |

The posting list of Arquivo.pt contains five fields:

- content
- title
- anchor
- url
- host

5.1.2 Workflow

1. The index-collection.sh creates an auxiliary DB with all URLs and sets up the Hadoop cluster for supporting collections types with multiple URLs (only for FAWP). This guarantees one Hadoop job per URL.

2. The index-collection.sh takes all the URLs from the DocumentServer and adds them to the crawldb. As a central part of Nutch, the crawldb maintains information on all known URLs.

3. Based on the data of crawldb, index-collection.sh creates a fetchlist and places it in a newly created segment directory.
4. The fetcher gets the content of the URLs on the fetchlist and writes it back to the segment directory. This step usually is the most time-consuming one.

5. Before indexing, all the links need to be inverted, the inverted links are saved in the linkdb.

6. Using data from all possible sources (crawldb, linkdb and segments), the indexer creates an index and saves it within the index directory.

7. Pruning Indexes, the posting lists of the five fields (content, title, anchor, url and host) are pruned. All documents on the posting lists of mime types not searchable are discarded. Fields stored not on index can also be discarded using the "-del" option followed by the field names. Execute the command without parameters to see the help description.

5.2 Installing the *Arquivo.pt* search system

This section joins all needed instructions for setting up the *Arquivo.pt* search system.

- required software and file system architecture appendix A
- guidance to set up the search system [24] [25]
- code location [26]
- code compilation [27]

5.3 Deploying a collection

This section provides a guidance about the process of deploying a collection to the search service.

Fig. 5.2 shows how the process of deploying a collection evolves, starting from the stored indexes until it gets accessible. [28] [29] [30] [31]

1. Setting up the QueryServer

Table 5.3 contains the bundle of files/folders for a QueryServer installation.
2. Setting up the Broker

Add a new entry in deploy/config/search-server.txt with the new collection, in this case AWP5.

Table 5.4 is an example of a file configuration for Broker.

Table 5.4: Example of Broker: search-servers.txt

<table>
<thead>
<tr>
<th>Machine_Name</th>
<th>Port_Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>p57.arquivo.pt</td>
<td>10015</td>
</tr>
</tbody>
</table>
Table 5.3: Bundle of files necessary for deploying indexes (Broker).

<table>
<thead>
<tr>
<th>Path</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>/deploy/configs/search-servers.txt</td>
<td>contains the location of the indexes supplied by each QueryServer.</td>
</tr>
<tr>
<td>/deploy/current/</td>
<td>contains the location of the web applications.</td>
</tr>
<tr>
<td>/arcproxy/</td>
<td>contains the arcproxy data base.</td>
</tr>
<tr>
<td>/dictionaries/</td>
<td>contains the indexes built for the query suggestion.</td>
</tr>
<tr>
<td>/run/</td>
<td>contains the tomcat pid file.</td>
</tr>
<tr>
<td>/scripts/</td>
<td>contains the binary to generate the arcproxy DB.</td>
</tr>
</tbody>
</table>

3. Test Broker

For checking up that the indexes were properly set up, you have to query the system with a full-text or URL search and the search results list should be returned. Figure 5.3 illustrates the search results list for the queries fecn and fecn.pt respectively.

Figure 5.3: Search results list when the indexes were properly set up.

4. Setting up the DocumentServer: HTTP accessibility

Figure 5.4: DocumentServer flowchart for deploying a collection

(a) Create symbolic links to arcfiles into tomcat webapp browser.

(b) Start tomcat;
5. Setting up the DocumentServer: ArcProxy database

(a) Guarantee that the collection is accessible through a HTTP:8080, i.e
\[ \text{p70.arquivo.pt:8080/browser/files/AWP5} \]

(b) Delete old files in \textit{collection/arc\_file\_AWP5.txt} that might contain previous configurations;

Table 5.5: Bash script to set up the ArcProxy

<table>
<thead>
<tr>
<th>Script name</th>
<th>Collection path</th>
<th>http request for browser.war</th>
<th>Broker server</th>
</tr>
</thead>
<tbody>
<tr>
<td>scripts/arcproxy.sh</td>
<td>-d collections/AWP5/</td>
<td>-u <a href="http://p70.arquivo.pt:8080/">http://p70.arquivo.pt:8080/</a></td>
<td>p58.arquivo.pt</td>
</tr>
</tbody>
</table>

6. Test Broker

For assessing the set up of the DocumentServer, you have to request the replay of a web content. Fig. 5.5 is an example of the page replay for the archived web content \[ \text{http://arquivo.pt/wayback/20140930090138/http://www.fccn.pt/pt/index.php} \].

5.3.1 Understanding the Broker search-servers.txt

This section underlines that the entry order on search-servers.txt have to be coherent by means of a cumbersome mistakes. Table 5.6 details an example of a search-servers.txt on a Broker.

Table 5.6: Excerpt from search-servers.txt on P58.arquivo.pt (Broker).

<table>
<thead>
<tr>
<th>QueryServer</th>
<th>Port_Name</th>
<th>Index Position</th>
<th>Collection Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>p55.arquivo.pt</td>
<td>21111</td>
<td>0</td>
<td>IA</td>
</tr>
<tr>
<td>p55.arquivo.pt</td>
<td>21112</td>
<td>1</td>
<td>Roteiro</td>
</tr>
<tr>
<td>p55.arquivo.pt</td>
<td>21113</td>
<td>2</td>
<td>BN</td>
</tr>
<tr>
<td>p55.arquivo.pt</td>
<td>60008</td>
<td>7</td>
<td>AWP8</td>
</tr>
<tr>
<td>p56.arquivo.pt</td>
<td>10011</td>
<td>8</td>
<td>AWP11</td>
</tr>
<tr>
<td>p56.arquivo.pt</td>
<td>10012</td>
<td>9</td>
<td>AWP12</td>
</tr>
<tr>
<td>p57.arquivo.pt</td>
<td>10001</td>
<td>19</td>
<td>AWP1</td>
</tr>
<tr>
<td>p57.arquivo.pt</td>
<td>10003</td>
<td>20</td>
<td>AWP3</td>
</tr>
<tr>
<td>p57.arquivo.pt</td>
<td>10004</td>
<td>21</td>
<td>AWP4</td>
</tr>
</tbody>
</table>

Table 5.7 details an example of a search-servers.txt on a QueryServer.
Hence, collection named IA is the first entry of search-servers.txt (Table 5.6), which means that on search-servers.txt (Table 5.7) the entry must also be the first.

Table 5.8 details a search-server.txt which does not match with the search-servers.txt on table 5.6. The first position on QueryServer is the collection Roteiro and on Broker is collection IA.

As a result, we could be displayed a misplaced web content. Fig 5.6 is an
5.4 Removing archived web content from the search

Arquivo.pt has two approaches for removing an index from the search result list.

Forbid access to archive web content: Apache HTTP redirects. Still, Apache HTTP only sets the index inaccessible to user, whereby it will be visible in the search result lists. Fig. 5.7 illustrates the returned Apache page to users;

Remove from the search results list: removes the access to a exact Lucene DocID. It sets the index to null, therefore there are not a matching index to a web content. Whereupon, the web content will not be displayed either
on the search results list or in format of web content. The web content is removed from both search results list shown on fig. 5.3 (32) (33);

5.5 Full-text search with Luke

Arquivo.pt made use of version 0.9.1 of Luke to do a full-text search on the generated indexes. Luke is a handy development and diagnostic tool, which accesses already existing Lucene indexes and allows you to display and modify their content.

The following describes how to search by URL using Luke.

1. Download Luke. (34)


Fig. 5.8 presents and example of an user invoking Luke.

Figure 5.8: Screenshot of first page of Luke

3. Select the path where the indexes are stored.

4. Select tab Search

5. Select the field exacturl

6. Get the expanded and encoded URL using opensearch API.


www.sapo.ua.pt/
7. Fig. 5.9 presents the chosen exact url: TFIEPHSBIPUFVRX2VD6RZJXE.

Figure 5.9: Opensearch header result for the query http://sapo.ua.pt.

8. On tab Analysis select KeywordAnalyser

9. On tab Hitcollector change to: "Return all matching results, even low-scored"

10. On the top right side insert: exacturl:TFIEPHSBIPUFVRX2VD6RZJXE

11. Press search and the result will printed out. The fig 5.10 presents the final result.

Figure 5.10: Searchinf for the URL http://sapo.ua.pt with Luke.

5.6 Installing the informative site (Arquivo.pt_CMS)

For a newly installation of Plone there are many plugins that are necessary to be installed and configured. (35) (36) (37)

The sobre.arquivo.pt installation of Plone, contains several scripts that are running in background, which are:
zopePloneApacheBackup.sh backs up the sobre.arquivo.pt every day and replicates the master version to its slave.

verificapagina.sh warns Arquivo.pt team when there are changes at sobre.arquivo.pt.

5.7 Monitoring the Arquivo.pt service

Arquivo.pt has the following tools to guarantee the best service quality.

Nagios: (Internal) monitors the infrastructure. [sirens.fccn.pt/nagios/](sirens.fccn.pt/nagios/)

Uptimerobot: (External) monitors the infrastructure. [https://uptimerobot.com/dashboard#mainDashboard](https://uptimerobot.com/dashboard#mainDashboard)


Google analytics: tracks and reports the Arquivo.pt and sobre.arquivo.pt traffic. [analytics.google.com/analytics/web/?authuser=0#home/a21825027w43114507p47220271/](analytics.google.com/analytics/web/?authuser=0#home/a21825027w43114507p47220271/)


Ganglia: is a scalable distributed system monitor tool for high-performance computing systems such as clusters and grids. [http://stats.arquivo.pt](http://stats.arquivo.pt)
Chapter 6

Customizing Full-text of Search Ranking

The ranking system orders by relevance millions of documents matching a given query. It uses several heuristics to compute a relevance value for each document matching a query, and presents the documents sorted by these values (40). On the Internet, users expect no more than a few seconds for a response. Hence, several ranking functions were added, such as query-dependent, query-independent and time-aware (42). Fig. 6.4 is the user interface developed for testing combinations of features and heights. For collaborators of Arquivo.pt it is available in \url{http://arquivo.pt/searchTests.jsp}. 
Figure 6.1: Webpage provided for testing ranking features
6.1 Query-dependent features

Query-dependent features depend both on the contents of the document and the query. Query-dependent models estimate the document’s relevance according to a given query.

Figure 6.2: searchTests.jsp: Query-dependent features.
6.1.1 Term-weighting functions

Term-weighting functions features estimate the similarity between the query and the different sections of a document version (anchor text of incoming links, text body, title, and URL) \( \text{[11]} \).

**Term Frequency (TF)** Term Frequency looks at term \( t \) in document \( d \), which counts the number of times that \( t \) occurs in \( d \). Suppose we have a collection of \( N \) documents. Define \( \{f_{i,j}\} \) to be the frequency (number of occurrences) of term (word) \( i \) in document \( j \). Then, the term frequency \( \{TF_{i,j}\} \) is defined as:

\[
TF_{i,j} = \frac{f_{i,j}}{\max_k f_{k,j}} \quad (6.1)
\]

That is, the term frequency of term \( i \) in document \( j \) is \( f_{i,j} \) normalized by dividing it by the maximum number of occurrences of any term (perhaps excluding stop words) in the same document. Thus, the most frequent term in document \( j \) gets a TF of 1, and other terms get fractions as their term frequency for this document \( \text{[51]} \).

**Inverse Document Frequency (IDF)** The IDF component acts to discriminate between informative and non-informative query terms. Those terms that have a high IDF are considered more informative, because they rarely occur in the collection. On the other hand, terms that have a low IDF are considered uninformative, since they occur in many documents. As the number of documents in a collection increases, IDF becomes increasingly important in order to discriminate between those documents that contain non-informative query terms and those that contain high informative query terms \( \text{[51]} \).

The IDF for a term is defined as follows. Suppose term \( i \) appears in \( \{n_i\} \) of the \( N \) documents in the collection. Then

\[
IDF_i = \log\left(\frac{N}{n_i}\right) \quad (6.2)
\]

TF-IDF is the combination of this two functions. Consider a document containing 100 words wherein the word *Arquivo.pt* appears 3 times. The
term frequency (i.e., tf) for Arquivo.pt is then \( \frac{3}{100} = 0.03 \). Now, assume we have 10 million documents and the word Arquivo.pt appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as \( \log(10,000,000 / 1,000) = 4 \). Thus, the TF-idf weight is the product of these quantities: \( 0.03 \times 4 = 0.12 \). [38]


```java
public PwaTFxIDF(Vector<Integer> tf, Vector<Integer> idf, int nTerms, int nDocs) {
    double tfNorm;
    double idfNorm;
    for (int i = 0; i < tf.size(); i++) {
        if (tf.get(i) != 0) {
            tfNorm = (double) tf.get(i);
            idfNorm = Math.log((double) nDocs / idf.get(i));
            score += tfNorm * idfNorm;
        }
    }
}
```

**BM-25** The relevance weighting model, also known as RSJ by the name of its creators (Roberston and Sparck-Jones), has been one of the most influential model in the history of Information Retrieval. It is a probabilistic model of retrieval that tries to answer the following question:

> What is the probability that this document could be relevant to this query?

'Query' is a particular instance of an information need, and 'document' a particular content description. The purpose of this question is to rank the documents in order of their probability of relevance according to the Probability Ranking Principle [39].


```java
public PwaBM25(Vector<Integer> tf, Vector<Integer> idf, int nTerms, double avgNTerms, int nDocs, double k1Parameter, double bParameter) {
    double tfNorm;
    double idfNorm;
    for (int i = 0; i < tf.size(); i++) {
        if (tf.get(i) != 0) {
            idfNorm = Math.log10((double) nDocs - idf.get(i) + 0.5) / (idf.get(i) + 0.5);
            tfNorm = (double) (tf.get(i) + k1Parameter) / ((tf.get(i) + k1Parameter) * (k1Parameter + (1 - bParameter) + bParameter * ((double) nTerms / avgNTerms)));
            score += idfNorm * tfNorm;
        }
    }
}
```
Lucene combines Boolean model (BM) of Information Retrieval with Vector Space Model (VSM) of Information Retrieval - documents "approved" by BM are scored by VSM.

Algorithm 6.3: Lucene Similarity (6)

```java
public PwaLuceneSimilarity(Vector<Vector<Integer>> tfPerField, Vector<Vector<Integer>> idfPerField, Vector<Integer> nTermsPerField, int nDocs) {
    double tfNorm;
    double idfNorm;
    Vector<Integer> tf;
    Vector<Integer> idf;
    int nTerms=0;
    double sumOfSquaredWeights=0;
    for (int j=0; j<tfPerField.size(); j++) {
        tf=tfPerField.get(j);
        idf=idfPerField.get(j);
        nTerms+=nTermsPerField.get(j);
    }
    for (int i=0; i<tf.size(); i++) {
        if (tf.get(i)!=0) {
            tfNorm=Math.sqrt((double)tf.get(i));
            idfNorm=1+Math.log((double)nDocs/(double)(idf.get(i)+1));
            sumOfSquaredWeights+=Math.pow(idfNorm, 2);
            idfNorm=Math.pow(idfNorm, 2);
            score+=tfNorm*idfNorm*norm(PwaIndexState.FIELDS[j], nTerms, boosts[j]);
        }
    }
    score=queryNorm(sumOfSquaredWeights);
}
```

Algorithm 6.4: Lucene Similarity Normalized (7)

```java
public PwaLuceneSimilarityNormalized(Vector<Vector<Integer>> tfPerField, Vector<Vector<Integer>> idfPerField, Vector<Integer> nTermsPerField, int nDocs) {
    PwaLuceneSimilarity similarity=tfPerField, idfPerField, nTermsPerField, nDocs);
    score=similarity.score();
    score=score*(float)Math.pow(Math.E, -1*score/MAX_SCORE);
}
```

Nutchwax Lucene but with a different normalization by field length (44).

Algorithm 6.5: Nutchwax Similarity (8)

```java
public PwaNutchSimilarity(Vector<Vector<Integer>> tfPerField, Vector<Vector<Integer>> idfPerField, Vector<Integer> nTermsPerField, int nDocs) {
    double tfNorm;
    double idfNorm;
    Vector<Integer> tf;
```
Vector<Integer> idf;
int nTerms=0;
double sumOfSquaredWeights=0;

for (int j=0; j<tfPerField.size(); j++) {
    tf=tfPerField.get(j);
    idf=idfPerField.get(j);
    nTerms=nTermsPerField.get(j);
    for (int i=0; i<tf.size(); i++) {
        if (tf.get(i)!=0) {
            tfNorm=Math.sqrt((double)tf.get(i));
            idfNorm=1+Math.log((double)nDocs/(double)(idf.get(i)+1));
            sumOfSquaredWeights+=Math.pow(idfNorm, 2);
            idfNorm=Math.pow(idfNorm, 2);
            score+=tfNorm*idfNorm*norm(PwaIndexStats.FIELDS[j], nTerms, boosts[j]);
        }
    }
}

score= queryNorm(sumOfSquaredWeights);

Algorithm 6.6: Nutchwax Similarity Normalized

public PwaNutchSimilarityNormalized(Vector<Vector<Integer>> tfPerField,
Vector<Vector<Integer>> idfPerField, Vector<Integer> nTermsPerField, int nDocs) {
    similarity=new PwaNutchSimilarity(tfPerField, idfPerField, nTermsPerField, nDocs);
    score=similarity.score();
    score=MAX_SCORE-score;
    score=(float)Math.pow(Math.E, -1*score/MAX_SCORE);
}

6.1.2 Term-distance functions

Term-distance features use the distance between terms in the different sections of a document version to quantify the relatedness between them, such as the Minimal Span Weighting function [41]. Further information available in https://github.com/arquivo/pwa-technologies/blob/master/PwaLucene/src/main/java/org/apache/lucene/search/PwaPositionsManager.java.

Span is defined as the length of the shortest document segment that covers all query term occurrences in a document, including repeated occurrences [52].

Minimum coverage (MinCover) MinCover is defined as the length of the shortest document segment that covers each query term at least once in a document [52].
Minimum Pair Distance (MinDist)  The minimum pair distance is defined as the smallest distance value of all pairs of unique matched query terms. Formally,

$$MinDist = \min_{q_1, q_2 \in Q \cap D, q_1 \neq q_2} \{Dis(q_1, q_2; D)\} \quad (6.3)$$

Average Pair Distance (AveDist) The average pair distance is defined as the average distance value of all pairs of unique matched query terms. Formally,

$$AveDist = \frac{2}{n(n-1)} \sum_{q_1, q_2 \in Q \cap D, q_1 \neq q_2} \{Dis(q_1, q_2; D)\} \quad (6.4)$$

, where $n$ is the number of unique matched query terms in $D$, and in the sum, we count $Dis(q_1, q_2; D)$ and $Dis(q_2, q_1; D)$ only once [52].

Term-distance ranking features are based on the above.

MinSpanCovUnord (Minimum Span Coverage Unordered) minimum span including all query terms.

MinSpanCovOrd (Minimum Span Coverage Ordered) minimum span including all query terms ordered as submitted.

MinSpanPairDist (Minimum Span Pair Distance) minimum distance between two query terms.

Algorithm 6.7: Computing Distances [10]

```java
public void computeDistances(int doc) throws IOException {
    minSpanCovUnordered=Integer.MAX_VALUE;
    minSpanCovOrdered=Integer.MAX_VALUE;
    minPairDist=Integer.MAX_VALUE;

    if (terms==null || terms.size()<2) {
        minSpanCovUnordered=0;
        minSpanCovOrdered=0;
        minPairDist=0;
        return;
    }

    for (int i=0; i<terms.size(); i++) {
```
positions[i]=terms.get(i).getPos(doc);
if (positions[i]==null) {
    return;
}
}

queue.clear();
int end = 0;
boolean done = false;
for (int i=0;i<positions.length;i++) {
    if (!positions[i].next()) {
        return;
    }
    if (positions[i].get()>end) {
        end=positions[i].get();
    }
    queue.put(positions[i]);
}
do {
    PwaPositions pos = (PwaPositions)queue.pop();
    int start = pos.get();
    int next = ((PwaPositions)queue.top()).get();
    for (int i=start;i<=next&&!done;) {
        start = i;
        if (!pos.next()) {
            done=true;
        }
        else {
            i=pos.get();
        }
    }
}

int matchLength = end-start - nTermsInQuery+1;
if (minSpanCovUnordered>matchLength) {
    minSpanCovUnordered=matchLength;
}
if (minSpanCovOrdered>matchLength) {
    boolean testOrder=true;
    for (int i=1;i<positions.length&&!testOrder;i++) {
        int a = (pos==positions[i-1]) ? start : positions[i-1].get();
        int b = (pos==positions[i]) ? start : positions[i].get();
        if (a>b || (offsetTerms==null & offsetTerms.get(i)-offsetTerms.get(i-1)==b-a)) {
            testOrder=false;
        }
    }
    if (testOrder) {
        minSpanCovOrdered=matchLength;
    }
}
if (minPairDist>next-start-1) {
    minPairDist=next-start-1;
}
if (minSpanCovOrdered==0) {
    done=true;
}
if (!done) {
    if (pos.get()>end) {
        end=pos.get();
    }
    queue.put(pos);
}
while (!done);
}

Algorithm 6.8: Minimum Span

```java
public PwaMinSpan(int span) {
    score = (float) Math.log(1 + Math.pow(Math.E, -1 * span));
}
```

6.2 Query-Independent Features

Query-Independent Features, depend only on the document, but not on the query. Therefore, these models rank documents according to an importance, quality or popularity measure computed independently of the query.

Figure 6.3: searchTests.jsp: Query-independent features.

**URL** features compute an importance measure based on the probability of URLs representing an entry page, using the number of slashes, their length, or if they refer to a domain, sub-domain or page.

**Web-graph** features estimate the popularity or importance of a document version inferred from the graph of hyperlinks between versions. These features include the number of inlinks to a version.

The following is the explanation of each query-independent feature:

**Url Depth** A URL defines the unique location of a document on the WWW. It consists on the name of the server, a directory path and a filename. A first observation is that documents that are right at the top level of a specific server (i.e. the documents of which the URL is no more than a
server name) are often entry pages. Also, as we descend deeper into the directory tree, the relative amount of entry pages decreases. Thus, the probability of being an entry page seems to be inversely related to the depth of the path in the corresponding URL. (47)

Algorithm 6.9: Url Depth

```java
public PwaUrlDepth(String url) throws MalformedURLException {
    URL url = new URL(url);
    String urlParts[] = url.split("/");
    int c = 2; // http + '
    if (urlParts.length==1+c) {
        score = 3;
        return;
    }
    if (urlParts[urlParts.length-1].indexOf('.')!=-1) { // file
        score = 0;
        return;
    }
    if (urlParts.length==2+c) {
        score = 2;
        return;
    }
    score = 1;
}
```

Url Slashes number of URL slashes. (47)

Algorithm 6.10: Url Slashes

```java
public PwaUrlSlashes(String url) throws MalformedURLException {
    score = 0;
    for (int i = 7; i < url.length(); i++) {
        if (url.charAt(i)=='/') {
            score += i;
        }
    }
}
```

Inlinks A hyperlink is a connection between a source and a target document. Our inlinks based prior is based on the observation that entry pages tend to have a higher number of inlinks than other documents (i.e. they are referenced more often). (47)

LinInlinks a Linearization of the number of inlinks. (47)

Algorithm 6.11: LinInlinks

```java
public PwaLinInlinks(int nInlinks) {
    if (nInlinks==0) {
        score = 0;
    } else {
        score = (float)Math.log10(Math.pow(nInlinks, POWER));
    }
}
```
6.3 Temporal features

The temporal features consider the time dimension of the web. They include the age of a document version and the features about lifespan based on the long-term persistence of web documents.

Figure 6.4: searchTests.jsp: Temporal features.

<table>
<thead>
<tr>
<th>Temporal features</th>
</tr>
</thead>
<tbody>
<tr>
<td>56 QueryIssueTime (in days)</td>
</tr>
<tr>
<td>57 Age - from query time (in days)</td>
</tr>
<tr>
<td>58 TimestampVersion (in days)</td>
</tr>
<tr>
<td>59 TimestampOldestVersion (in days)</td>
</tr>
<tr>
<td>60 TimestampNewestVersion (in days)</td>
</tr>
<tr>
<td>61 SpanVersions (in days)</td>
</tr>
<tr>
<td>62 SpanVersions (normalized)</td>
</tr>
<tr>
<td>63 NumberVersions</td>
</tr>
<tr>
<td>64 NumberVersions (normalized)</td>
</tr>
<tr>
<td>65 BoostNewer</td>
</tr>
<tr>
<td>66 BoostOlder</td>
</tr>
<tr>
<td>67 BoostNewerAndOlder</td>
</tr>
</tbody>
</table>

**QueryIssueTime** Query issue time in days

**Age - from query time** Timespan in days from the query issue time to the version date

Algorithm 6.12: Age

```java
public float PwaAge(long docTimestamp, long queryTimestamp) {
    float diff = queryTimestamp - docTimestamp;
    score = diff / DAY_MILLISEC;
}
```

**TimestampVersion** Age of the version in days

**TimestampOldestVersion** Age of the oldest version of the same URL in days

**TimestampNewestVersion** Age of the newest version of the same URL in days
**SpanVersions**) Days between the oldest and newest version of the same URL

Algorithm 6.13: Java code for SpanVersions feature

```java
public PwaSpanVersions(long maxTimestamp, long minTimestamp, long maxSpan) {
    long diff = (maxTimestamp - minTimestamp) / (long)DAY_MILLISEC;
    if (maxSpan == 0) {
        score = 0;
    } else if (diff == 0) {
        score = 0;
    } else {
        score = (float)Math.log10(diff) / (float)Math.log10(maxSpan);
    }
}
```

**SpanVersions (normalized)** Normalized days between the oldest and newest version of the same URL

**NumberVersions** Number of versions of the same URL

Algorithm 6.14: NumberVersions

```java
public PwaNumberVersions(long numberVersions, long maxNumberVersions) {
    score = (float)Math.log10(numberVersions) / (float)Math.log10(maxNumberVersions);
}
```

**NumberVersions (normalized)** Normalized number of versions of the same URL

**BoostNewer** Exponential decay of the age of the version that boosts more recent versions

**BoostOlder** Exponential decay of the age of the version that boosts more older versions

Algorithm 6.15: BoostOlder

```java
public PwaBoostOlder(long docTimestamp, long maxTimestamp, long minTimestamp) {
    float maxSpan = maxTimestamp - minTimestamp;
    maxSpan /= DAY_MILLISEC; // span in days
    float span = docTimestamp - minTimestamp;
    span /= DAY_MILLISEC; // span in days
    score = (float)Math.pow(Math.E, -1 * span / maxSpan);
}
```

**BoostNewerAndOlder** Exponential decay of the age of the version that boosts more recent and older versions
Algorithm 6.16: BoostNewerAndOlder  

```java
public PwaBoostNewerAndOlder(long docTimestamp, long maxTimestamp, long minTimestamp) {
    long middle=(maxTimestamp+minTimestamp)/2;
    if (docTimestamp<middle) {
        PwaBoostOlder ranker=new PwaBoostOlder(doc Timestamp, maxTimestamp, minTimestamp);
        score=ranker.score();
    } else {
        PwaBoostNewer ranker=new PwaBoostNewer(docTimestamp, maxTimestamp, minTimestamp);
        score=ranker.score();
    }
}
```

6.4 Ranking system used in the Arquivo.pt

Arquivo.pt developed a ranking system based on learning to rank algorithm in order to find the best ranking model. The project is based on Lucene, but with improvements for instance on the Lucene features. PwaLucene is available in https://github.com/arquivo/pwa-technologies/tree/master/PwaLucene. The file wax-default.xml defines the weights for each ranking feature in production. The following frame illustrates the file wax-default.xml in production (https://github.com/arquivo/pwa-technologies/blob/master/PwaArchive-access/projects/nutchwax/conf/wax-default.xml).

```xml
<name> ranking.functions </name>
<value>34 0.023 49 0.59 37 0.345 45 1.259</value>
<description>
</description>
```

The features are imported to the system through https://github.com/arquivo/pwa-technologies/blob/master/PwaLucene/src/main/java/apache/lucene/search/PwaFunctionsWritable.java and afterwards this functions will be sent over protocol RPC for ranking documents. The backbone of the L2R is located in https://github.com/arquivo/pwa-technologies/blob/master/PwaLucene/src/main/java/apache/lucene/search/PwaScorerFeatures.java where the whole process of ranking is processed.

The fig. 6.5 illustrates the searchTests.jsp set up with the actual features of Arquivo.pt. This webpage page provides an interface for assessing all the
combinations of *Arquivo.pt* features.

Figure 6.5: Boosting new features

The table 6.1 details the ranking features that are in production.

### Table 6.1: Ranking features in production in *Arquivo.pt*

<table>
<thead>
<tr>
<th>Feature number</th>
<th>Name of the features</th>
<th>Fields</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>Nutch</td>
<td>content + url + host + anchor + title</td>
<td>0.023</td>
</tr>
<tr>
<td>49</td>
<td>MinSpanCovUnord</td>
<td>title</td>
<td>0.593</td>
</tr>
<tr>
<td>37</td>
<td>MinSpanCovUnord</td>
<td>content</td>
<td>0.345</td>
</tr>
<tr>
<td>45</td>
<td>MinSpanCovOrd</td>
<td>anchor</td>
<td>1.259</td>
</tr>
</tbody>
</table>

### 6.5 Future Work

*Arquivo.pt* has developed a test collection named PWA9609 ([https://github.com/arquivo/pwa-technologies/wiki/TestCollection](https://github.com/arquivo/pwa-technologies/wiki/TestCollection)). A L2R dataset for researching learning to rank applied to WAIR was created. This dataset named L2R4WAIR, more information available in [https://github.com/arquivo/pwa-technologies/wiki/L2R4WAIR](https://github.com/arquivo/pwa-technologies/wiki/L2R4WAIR). The top 6 most important ranking features which were achieved for the temporal-dependent are:

- BM25 over all fields
- TD-IDF over all fields
- Number of versions of a URL
- TF-IDF over the hostname of URL
- Length of the shortest text with all query terms in title
- Days between the first and last versions of a URL
Chapter 7

Arquivo.pt APIs

The Arquivo.pt research infrastructure allows automatic access, both for URL and full-text search queries, through its Memento Time Travel and OpenSearch APIs.

7.1 Memento Time Travel API

Time Travel service helps you find and view versions of Web pages that existed at some time in the past.

These prior versions of web pages are named Mementos. Mementos can be found in web archives or in systems that support versioning such as wikis and revision control systems. (50)

PWA-Technologies software supports Memento Time travel API natively, because it uses pywb software as the wayback for the replay of archived Web-pages.

Arquivo.pt has the following memento configurations:

- Timegate: http://arquivo.pt/wayback/
- Timemap: http://arquivo.pt/wayback/timemap/*/

For more informations about Arquivo.pt TimeTravel API please visit the following links:


Memento Time Travel API only allows URL Search.

For automatic access to the full-text search results one should use the OpenSearch API.

### 7.2 OpenSearch API

*Arquivo.pt provides an interface for users and tools to easily query the system. The response is a XML-based file (RSS 2.0).*

Arquivo.pt OpenSearch API allows automatic full-text search and URL search. Unlike when using Memento Timetravel API, where one can make searches in multiple World Web archiving initiatives, when using the OpenSearch API, the queries are done exclusively over the Arquivo.pt content.

For more informations about Arquivo.pt OpenSearch API please visit the following links:

- **OpenSearch 1.1 (Draft 5) namespace:** [http://www.opensearch.org/Specifications/OpenSearch/1.1/Draft_5](http://www.opensearch.org/Specifications/OpenSearch/1.1/Draft_5)


Chapter 8

PyWb

*PyWb is a Python implementation of Web archival replay tools, sometimes also known as Wayback Machine* \cite{48}.

As stated in the previous chapters, Arquivo.pt uses the .ARC file format to store the archived Web content, and custom Lucene indexes for searching. These custom Lucene indexes are also used for the replay of the Web pages. PyWb doesn’t support Arquivo.pt’s custom Lucene indexes natively, however it is possible to use pyWb with Arquivo’s indexes, thanks to the pywb-opensearch-cdx wrapper \cite{49}, developed by the creator of pyWb, that captures the information through the OpenSearch API, and converts the results to CDXs.

The pyWb module, adapted for the replay of Arquivo.pt, is available in [github](https://github.com/arquivo/pywb-opensearch-cdx/)

Two important files are the yaml files, namely (i) config.yaml - configurations file for the pyWb (replay); and (ii) unframed.yaml - configurations file for the pyWb unframed.

The config.yaml is shown in figure \ref{fig:configyaml}. One can observe in the line number 2 that pyWb is running in root http path (i.e. arquivo.pt/wayback/).

Line number 5 calls the wrapper that reads openSearch results and converts them to CDXs, i.e. it calls the class `OpenSearchCDXServer` in the file `opensearch.py`.

Line number 9 defines the *archive paths*, i.e. the location for the AR-C/WARC files. In the case of Arquivo.pt, these are the arcproxy http locations,
that will then redirect for the location of the Arc files.

Line number 13 defines that the cdx-server API is enabled. Line number 14 defines that the framed mode is enabled, i.e. the archived Web pages are contained in an iframe.

Lines 15, 16 and 17 define the paths for the static resources.

Finally, lines 18 and 19 define a property with the name search_location. This property is used by the html template responsible for the replay of archived Web pages.

Figure 8.2 shows a similar file, but for pyWb (unframed). The differences are: (i) unframed yaml runs with the path replay/ (i.e. arquivo.pt/noFrame/replay/); and (ii) framed_mode is set to false, so this version of pyWb is not contained in an iframe.

Figure 8.1: config.yaml for pyWb (replay)

Figure 8.3 shows the µWsgi configurations for the pyWb (replay) version.

Line 3 presents an http socket in the port 8585. This is used only for debugging.

In line 5 one can observe that there is a µWsgi socket on port 8282. This socket connects µWsgi and Apache Httpd, that loads a specific µWsgi module to enable this communication. The unframed version has a similar µWsgi ini file, but it does not have an http socket, and the µWsgi socket runs on port 8484.
Figure 8.2: unframed.yaml for pyWb (noFrame)

```yaml
unframed.yaml

collections:
  replay:
    index_paths: 'http://arquivo.pt/nutchwax/opensearch'
    server_cls: toolbar/name:opensearch.OpenSearchCDXServer

# configure path to warcs here
archive_paths:

enable_cdx_api: true
framed_replay: false
static_routes:
  static/resources: ./static/
  static/__pywb__: pywb/static/
metadata:
  search_location: http://arquivo.pt
```

Figure 8.4 shows the Apache Httpd configurations for the pyWb instances, and figure 8.5 shows how the pyWb instances are connected with µWsgi.

The pywb-opensearch-cdx project contains a folder named templates. This folder has several html templates that were adapted for Arquivo.pt Web infrastructure:

- `error.html` (custom page for 500’s http errors)
- `frame_insert.html` (replay page)
- `not_found.html` (custom page for 400’s http errors)
- `query.html` (page responsible for displaying url query results, currently empty and redirecting to nutchwax)
Figure 8.3: µWsgi.ini for pyWb (replay)

```ini
[uwsgi]
if-not-env = PORT
http-socket = :8080
socket = :8282
endif =
master = true
buffer-size = 65536
die-on-term = true

# gevent = 100
# gevent-early-monkey-patch =
# processes = 30

# specify config file here
env = PYTHON_CONFIG_FILE=conf.yml

env LANG="en_US.utf8"
env LC_ALL="en_US.UTF-8"
env LC_LANG="en_US.UTF-8"

wsgi = pywb.apps.wayback
```

Figure 8.4: µwsgi configurations in apache Httpd, located in /etc/httpd/vhosts.d/arquivo.conf

```xml
<Location /wayback>
    ProxyPass uwsgi://127.0.0.1:8282/
</Location>

<Location /noFrame>
    ProxyPass uwsgi://127.0.0.1:8484/
</Location>
```

Figure 8.5: µwsgi connections with apache Httpd.
Appendix A

Software Requirements

This appendix details the required files, applications and configurations for setting up the Arquivo.pt search system. Further information are available in https://github.com/arquivo/pwa-technologies/wiki/Install and http://wiki.priv.fccn.pt/Install_Search_Environment.

Requirements of Broker

/home/wayback/searcher/ : project root;
apache-tomcat-8.0.30/ : contains the Apache Tomcat binaries;
arcpoxy/ : contains the BarkeleyDB;
dictionaries/indexIA : contains the dictionaries for spellchecker.war;
deploy/current : contains web applications;
deploy/configs/ : contains the search-servers.txt;
run/ : contains the pids of running applications;
scripts/ : contains the arcpoxy.sh script;
Apache HTTP : Apache HTTP.

Requirements of DocumentServer

/opt/searcher/ : project root;
apache-tomcat-8.0.30/ : contains the Apache Tomcat binaries;
collections/ : contains collections that are on this DocumentServer;
run/ : contains the pids of running applications.

Requirements of QueryServer

/home/wayback/searcher/ : project root;

hadoop-search-servers : contains the HadoopRPC server;

run/ : contains the pids of running applications;

collections/ contains QueryServer configurations;

collections/blacklists/ contains Lucene index that are removed from search results;

../indexes/ contains indexes;

/etc/init.d/hadoopserver is the binary to start the QueryServer.

Version of the current Arquivo.pt installation:

NutchWax 0.11.0

PWA_Lucene 1.0.0 (extensions of lucene-2.1.0)

Wayback 1.2.1

Tomcat 8.0.30

Hadoop-DEV 0.14.5

Maven 3.x

ANT 1.7.1

JAVA SE 7.x

Plone 3.0.3

LinguaPlone 2.0

qPloneComments 3.1

PloneGazette 3.0.0
Appendix B

Mime-types indexed

The configuration file that contains information about the mime-types indexed is at https://github.com/arquivo/pwa-technologies/blob/master/PwaArchive-access/projects/nutchwax/nutchwax-webapp/target/nutchwax-webapp-0.11.0-SNAPSHOT/WEB-INF/classes/mime-types.xml.

The following list details which are the set up mime-types:

- application/andrew-inset
- application/javas
- application/mac-binhex40
- application/mac-compactpro
- application/msword
- application/oda
- application/dfp
- application/postscript
- application/smil
- application/vnd.mif
- application/vnd.ms-excel
- application/vnd.ms-powerpoint
image/gif
image/ief
image/jpeg
image/photoshop
image/png
image/tiff
image/vnd.djvu
image/vnd.wap.wbmp
image/xcf
image/x-cmu-raster
image/x-portable-anymap
image/x-portable-bitmap
image/x-portable-graymap
image/x-portable-pixmap
image/x-rgb
image/x-xbitmap
image/x-xpixmap
image/x-xwindowdump
message/news
message/rfc822
model/iges
model/mesh
model/vrml
text/css
Bibliography


features/querydependent/PwaLuceneSimilarityNormalized.java, 2016. [Online; accessed 04-Fev-2016].


Information Retrieval, SIGIR ’02, pages 27–34, New York, NY, USA, 2002. ACM.


85