LAB 7: Search engine: Apache Nutch + Solr + Lucene

Apache Nutch *Crawler + indexer (mainly crawler)*
Apache Lucene *indexer + searcher*
Apache Solr *indexer + searcher*

Lucene vs. Solr? Lucene = library, more low-level access, Solr = you can use it out-of-box.

1. **Apache Nutch:ApacheNutch** is an open software project for web crawling. The main characteristics of **ApacheNutch** are:

- **Nutch** is written in Java but the data is written in language-independent formats (XML,JSON, etc.). Thus, the data can be processed further.
- **Nutch** provides a transparent alternative to commercial web search engines.
- **Nutch** is highly extensible. It provides interfaces such as **Parse**, **Index**, or **ScoringFilters** for custom implementations. For instance, one may implement **Parse** interface and involve **Apache Tika** for document processing.
- **Nutch** has modular architecture (the core elements are **searcher**, **indexer**, **database**, **fetcher**) and is pluggable. Developers may create plug-ins for, e.g., data processing, data retrieval, scoring, etc., to tune **Nutch**.
- **Nutch** is highly scalable. It can be used at personal (e.g., index user's files, e-mails), local (e.g., intranets), or global (e.g., web browsers) scale.
- The project is split into two development directions:
    i. Nutch 1.x: uses Apache Hadoop data structures that are great for batch processing.
    ii. Nutch 2.x: uses Apache Gora for data store independence.
- http://www.cs.put.poznan.pl/mtomczyk/ir/lab7/CN-TR-04-04.pdf

2. **Apache Lucene** is an open source search engine library written in Java. The main features of **Apache Lucene** are:

- It is a high-performance search engine written in Java: low RAM requirements, fast incremental indexing, and small indexes.
- Provides tools for ranked and fielded searching. Documents are sorted according to relevance or any field (e.g., "title" or "author").
- **Lucene** supports various complex types of query, e.g., phrase queries, wildcard queries, proximity queries, or range queries.
- **Lucene** supports multiple-index searching and the results can be merged.
- **Lucene** provides advanced tools for aiding users, such as faceting, highlighting, joins and result grouping.
- **Lucene** can update database and allows searching at the same time.
- **Lucene** ranking models can be configured/tuned by using plugins.

3. **Apache Solr (pronounced Solar): ApacheSolr** is an open software search platform built on **Apache Lucene**. The main characteristics of **Apache Solr** are:

- **Solr** is written in Java from the **Apache Lucene** project.
- **Solr** is a web application.
- Since **Solr** is based on **Lucene**, it provides advanced matching like: phrases, wildcards, joins, grouping, etc. What is more, it provides full-text search, hit highlighting, or faceted search.
- Taking advantage of **Lucene**, **Solr** provides real-time indexing. Documents can be provided (indexed) via, e.g., JSON, XML, or CSV over HTTP.
- Query is given via HTTP GET and the results are received via, e.g., JSON, XML, or CSV.
- **Solr** provides administrative user interface for application monitoring and control. Many metrics are provided via JMX to get a better insight on performance.
- **Solr** has a pluggable architecture to support customization.
- **Solr** easily scales up or down and is fault tolerant.

Exercises

The purpose of the following exercise is to make you acquainted with Apache Nutch, Solr, and Lucene. These libraries, especially Nutch and Solr, are complex and rather difficult to configure. Consequently, the following exercises are rather simple. The organization of the two following two laboratories is as follows:

1) Lab 7: You will be working with pre-build Nutch (version 1.x) and Solr (you do not need to compile the sources). You will learn how to get Nutch and Solr to work together, i.e., learn the basic Nutch configuration (where it should crawl) and how to use Solr as the search tool. Lastly, you will make a simple Lucene Java application for searching. This application will use indexes generated by Nutch.
2) Lab 8: This lab is devoted to Apache Lucene. You will make an application for searching concerning: indexing, ranking models, and types of query.

Exercise 1. This exercise is based on https://wiki.apache.org/nutch/NutchTutorial.

1) Go to https://www.cs.put.poznan.pl/mtomczyk/ir/lab7/animals/. Your task is to obtain (crawl) the content of this directory and to construct a simple search engine. Display **Lemur.html**. Navigate from one page to another using the below links. Can you find a page devoted to a Hedgehog or a Parrot?
2) Download solr-6.6.0.zip and apache-nutch-1.14.zip. Put them together in any directory and extract the content. Run the console and move to the directory where Solr and Nutch are located.
3) Firstly, you can make some variables (paths):

```
nutch=`ls -d apache-nutch-1.14| tail -1`
echo $nutch
```

```
solr=`ls -d solr-6.6.0 | tail -1`
echo $solr
```

4) Then, create resources for a new Solr core. A core is a single index, a transaction log, and a set of configuration files. Solr can have multiple cores (indexes).

```
cp -r $solr/server/solr/configsets/basic_configs
$solr/server/solr/configsets/nutch
```

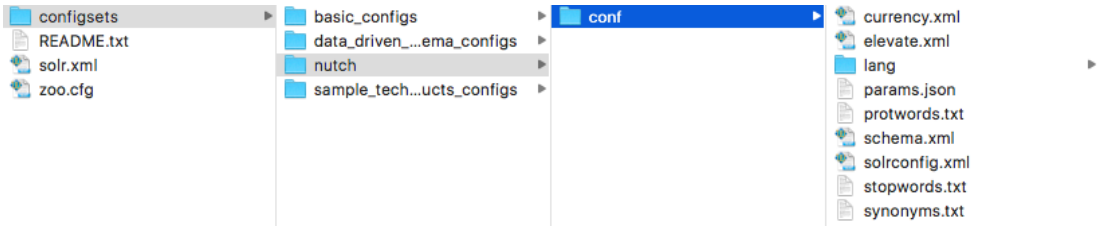
5) Create a configuration file. You can copy default configuration file (schema.xml) from the Nutch installation:

```
cp $nutch/conf/schema.xml
$solr/server/solr/configsets/nutch/conf
```

6) Remove managed-schema from the configuration folder:

```
rm $solr/server/solr/configsets/nutch/conf/managed-schema
```

You can verify the content of the configuration folder:



7) Now, start the Solr server:

```
./$solr/bin/solr start
```

You can stop it at any moment with (but do not do it now ☺):

```
./$solr/bin/solr stop -p 8983
```

8) Go to http://localhost:8983/solr/#/. You can see the administrative user interface. As you may notice, there is no core available.

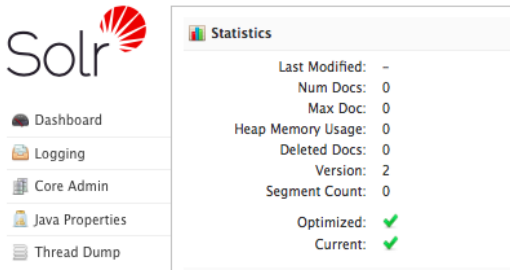
9) Create Nutch core based on the created configuration folder:

```
./$solr/bin/solr create -c nutch -d
$solr/server/solr/configsets/nutch/conf/
```

```
Creating new core 'nutch' using command:
http://localhost:8983/solr/admin/cores?action=CREATE&name=nutch&instanceDir=nutch

{
  "responseHeader":{
    "status":0,
    "QTime":9089},
  "core":"nutch"}
```

10) Check the administrative user interface (refresh). Navigate to Nutch core. Obviously, the index is empty (there are no documents).



11) Now, use Nutch to crawl the web and feed Sorl's index. Firstly, verify the Nutch installation:

```
./$nutch/bin/nutch
```

```
Usage: nutch COMMAND
where COMMAND is one of:
  readdb            read / dump crawl db
  mergedb           merge crawldb-s, with optional filtering
  readlinkdb        read / dump link db
```

12) If you have "permission denied" problem, you can run the **chmod** command:

```
chmod +x $nutch/bin/nutch
```

13) If you see that JAVA_HOME is not set, you have to set it:

```
export JAVA_HOME=path to java jdk,
for instance:
export JAVA_HOME=/usr/java/ some jdk /bin/java

For Mac, you can try:
export JAVA_HOME=$(/usr/libexec/java_home)
```

14) Now, configure Nutch. Do you remember http://www.cs.put.poznan.pl/mtomczyk/ir/lab7/animals/Lemur.html page? We want to crawl all pages under www.cs.put.poznan.pl/mtomczyk/ir/lab7/animals. As you may notice, each page (e.g., Lemur.html) contains one URL to Wikipedia at the bottom. We do not want Nutch to go there. Firstly, go to **conf** folder and open nutch-default-xml. This file contains a lot of parameters/properties that one may change to tune Nutch. However, you should not do so here. Open nutch-site.xml file. One may put there some properties to override these contained in nutch-default.xml. The (default) nutch-site.xml should look like the following:

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

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

<configuration>

</configuration>
```

15) To tune Nutch, add some properties to nutch-site.xml. Firstly, set the name of your crawler. You can set any name you want. Secondly, set fetcher.server.delay property to 0.5. This is the number of seconds the fetcher will delay between successive requests to the same server. The default value is 5.0 seconds. Since the crawler is expected to download pages located under http://www.cs.put.poznan.pl/mtomczyk/ir/lab7/animals and there are around 200 pages, 5.0s delay is definitely too long ☺.

```
<property>
<name>http.agent.name</name>
<value>IRbot</value>
</property>

<property>
<name>fetcher.server.delay</name>
<value>0.5</value>
```

```
</property>
```

16) Open regex-urlfilter.txt file. In this file, you can add URL filters in a form of regular expressions. Locate the following line:

```
# accept anything else
+.
```

Change it to the following two rules. The first rule ("-") filters out all URLs of Wikipedia pages. The second rule ("+") forces Nutch to accept URLs of pages under …/animals directory only (so the "-" rule is redundant ☺).

```
-^https://en\.wikipedia\.org/wiki/

# accept anything else
+^http://www\.cs\.put.poznan\.pl/mtomczyk/ir/lab7/animals
/
```

17) Now, create a list of seed URLs. Firstly create an "exercise" directory. It is expected that the outcomes of the crawler will be stored in this folder. In "exercise" create "urls" folder.

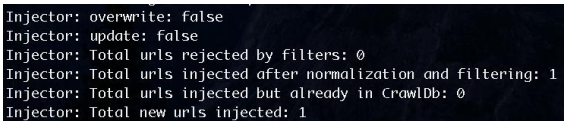
```
mkdir –p $nutch/exercise/urls/
```

Then, create seed.txt and add http://www.cs.put.poznan.pl/mtomczyk/ir/lab7/animals/Lemur.html URL to this file:

```
touch ./$nutch/exercise/urls/seed.txt
```

18) OK. Now you should use Nutch for crawling. The crawler starts from the Lemur page and is expected to derive, using extracted URLs, all pages under …/animals/ directory. What is more, the data will be processed to Solr that will build the index (Nutch core) and will allow retrieving information (searching). Firstly, run the **inject** command to pass the seed URL to **crawldb** (database of all known URLs):

```
./$nutch/bin/nutch inject $nutch/exercise/crawldb
$nutch/exercise/urls
```



Then, use **readdb** to read the database (**crawldb**) and store in …/dump1:

```
./$nutch/bin/nutch readdb $nutch/exercise/crawldb -dump
$nutch/exercise/dump1
```

Open dump1/part-00000 file and verify the content. Check if Lemur.html is contained in the database and check its status. It should be "db_unfetched" which means that the content of this page has not been downloaded yet.

```
http://www.cs.put.poznan.pl/mtomczyk/ir/lab8/animals/Lemur.html Version: 7
Status: 1 (db_unfetched)
Fetch time: Thu Mar 01 12:15:12 CET 2018
Modified time: Thu Jan 01 01:00:00 CET 1970
Retries since fetch: 0
Retry interval: 2592000 seconds (30 days)
Score: 1.0
Signature: null
Metadata:
```

19) Now, prepare a set of URLs that will be fetched next (obviously, only Lemur.html). Use **generate** command:

```
./$nutch/bin/nutch generate $nutch/exercise/crawldb
$nutch/exercise/segments
```

Generate command prepares a chunk of data to be processed, so-called **segment**. Check the ID of the generated segment (…/segments). Then, run **fetch** command to download the pages (SEG_ID is the ID of the generated segment). How many threads were used? How many pages were downloaded?

```
./$nutch/bin/nutch fetch $nutch/exercise/segments/SEG_ID
```

```
Fetcher: threads: 10
Fetcher: time-out divisor: 2
QueueFeeder finished: total 1 records + hit by time limit :0
FetcherThread 37 Using queue mode : byHost
FetcherThread 41 fetching http://www.cs.put.poznan.pl/mtomczyk/ir/lab8/animals/Lemur.html
queue crawl delay=500ms)
FetcherThread 37 Using queue mode : byHost
FetcherThread 42 has no more work available
FetcherThread 42 -finishing thread FetcherThread, activeThreads=1
FetcherThread 37 Using queue mode : byHost
FetcherThread 43 has no more work available
FetcherThread 43 -finishing thread FetcherThread, activeThreads=1
```

20) Now, parse the downloaded pages. Use **parse** command:

```
./$nutch/bin/nutch parse $nutch/exercise/segments/SEG_ID
```

```
Parsed (330ms):http://www.cs.put.poznan.pl/mtomczyk/ir/lab8/animals/Lemur.html
```

21) Use **readseg** to read the processed data:

```
./$nutch/bin/nutch readseg -dump
$nutch/exercise/segments/SEG_ID/ $nutch/exercise/dump2
```

Open dump2/dump file and verify its content. Can you find the content (text) of Lemur.html? Can you find the links extracted from Lemur.html? What is their status?

22) It is time to update the database (**crawldb**):

```
./$nutch/bin/nutch updatedb $nutch/exercise/crawldb
$nutch/exercise/segments/SEG_ID/
```

Then, read the database and verify the outcomes.

```
./$nutch/bin/nutch readdb $nutch/exercise/crawldb -dump
$nutch/exercise/dump3
```

What is the status of URL of Lemur.html? What about the rest of the URLs? Go to www.cs.put.poznan.pl/mtomczyk/ir/lab7/animals/Lemur.html. Did Nutch extract all links of Lemur.html (hint: one link is skipped ☺. Which one and why?)?

23) You may now feed the **linkdb** (database of **links)**.Use **invertlinks** command:

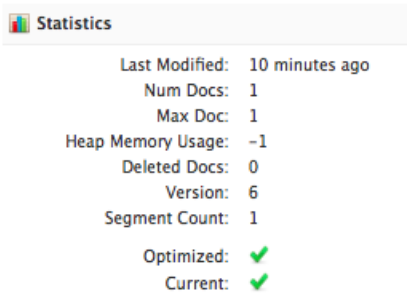
```
./$nutch/bin/nutch invertlinks $nutch/exercise/linkdb -
dir $nutch/exercise/segments
```

24) Now, it is time to feed **Solr**☺. Use **index** command. You have to pass the server address (-Dsolr.server.url=), the **crawldb**, the **linkdb**, and the location of generated segments:

```
./$nutch/bin/nutch index -
Dsolr.server.url=http://localhost:8983/solr/nutch
$nutch/exercise/crawldb -linkdb $nutch/exercise/linkdb
$nutch/exercise/segments/*
```

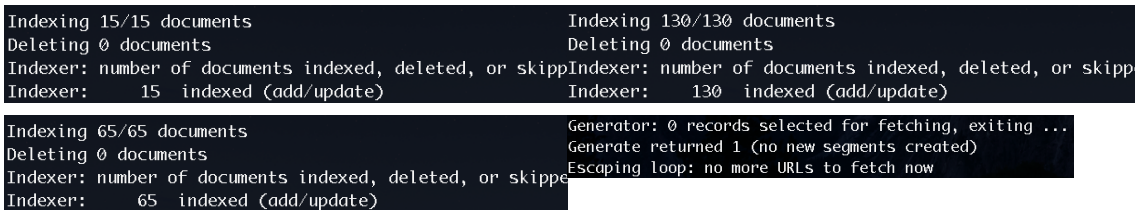
```
Indexing 1/1 documents
Deleting 0 documents
Indexer: number of documents indexed, deleted, or skipped:
Indexer:      1  indexed (add/update)
```

25) Go to http://localhost:8983/solr/#/ and verify whether Solr has indexed 1 page (Lemur.html):



26) Let's continue the crawling in order to obtain all pages and build the index. However, you do not have to repeat all the previous steps (generate → … → index) manually. You can use **crawl** command. The parameter "1" means that one round (iteration) of crawling will be performed. Do not change this parameter. Run this command until all pages are indexed (default configuration of Nutch makes that the indexed pages cannot be fetched again). How many pages were indexed after each of the rounds?

```
./$nutch/bin/crawl -i –
Dsolr.server.url=http://localhost:8983/solr/nutch -s
$nutch/exercise/urls/ $nutch/exercise/ 1
```



27) Go to http://localhost:8983/solr/#/ and check the number of indexed pages (211).

28) Go to the **query** page (for Nutch core). It is time to search for some information.

   a. Firstly, add *,score to **fl** variable in order to add **score** (the greater the score the more relevant the document) field to each returned document.
   b. Set **q** field = content:lemur. Search and verify the results. Which document has the greatest score? Compare the score of the 1st document with the score of the 2nd document. Is the difference small of great?
   c. You can build more complex queries. Try: content:lemur AND content:primates. How many documents were returned? Why? Compare the score of the 1st document with the previously obtained (point b) score of the 1st document. Which score is greater and why?
   d. Now, try title:lemur. How many documents are returned now?
   e. Try content:panda. Consider the first two documents. Are these documents relevant?
   f. Now, try content:turkey. What is the rank of turkey.html? Which document is the first? Justify this result. You can open these two pages in a browser and analyze the content. Are you surprised by the content of **turkey.html**? ☺

g. Try content:turkey AND content:country. How many documents are returned? Consider the 2nd document. Why is it on the list? Compare the scores of the returned documents and justify the difference.

Exercise 2. Your task is to build a simple Lucene application for searching. You should use the index that has been constructed by Solr. Solr uses Lucene to build an index so the formats are compatible. Firstly, you should create a Java project using any Java IDE. Then, go to the Solr folder and seek for the index. You should navigate to "server/solr/**nutch**/" and copy the **data** folder to the directory of Java project that you have created. Obviously, this is not the way one would like to integrate Nutch and Lucene ☺. However, because of the limited time and focusing on Lucene rather than Nutch-Lucene integration, such workaround is sufficient. Download **lucene-7.2.1.zip** and unzip it in the project's directory. Add the following jars to your project: **lucene-core-7.2.1.jar**, **lucene-queryparser-7.2.1.jar** and **lucene-backward-codecs-7.2.1.jar**.

29) Firstly, your application should load the index. Use **DirectoryReader.open()** to load the index from a file and use **IndexReader** to keep this index. When it comes to **DirectoryReader.open()**, you may use FSDirectory.open(Paths.get("data/index")) to get the directory. Do not forget to close the reader (the end of the code).
30) Construct **IndexSearcher** object. It can be used to search the index for documents that are relevant for a provided query.
31) Use **String** to initialize a query. Let it be "panda".
32) Use **StandardAnalyzer** (for text processing) and **QueryParser** to construct a **Query** object.
33) Use **TopDocs** objects and **search** method (IndexSearcher) to derive **n** top documents. Set $n = 5$.
34) **scoreDocs** (of TopDocs) contains IDs and scores of the derived documents.
35) For **n** top documents print: a title, a score, and content. To get document's content use **IndexSearcher** object and docID.