

Crawling with Apache Nutch

Nilanjan Debnath
MDS201919

Vanshi Mishra
MDS201939

Chennai Mathematical Institute
December 11, 2020

Introduction

A web crawler downloads and indexes content from all over the Internet. The goal of such a bot is to learn what (almost) every webpage on the web is about, so that the information can be retrieved when it's needed. They're called "webcrawlers" because crawling is the technical term for automatically accessing a website and obtaining data via a software program.

Apache Nutch is a web crawler software product that can be used to aggregate data from the web. It is used in conjunction with other Apache tools, such as Hadoop, for data analysis. We use Apache Nutch version 2.3 here for making a web crawler.

HBase is an open-source non-relational distributed database modeled after Google's Bigtable and written in Java. It is developed as part of Apache Software Foundation's Apache Hadoop project and runs on top of HDFS (Hadoop Distributed File System) or Alluxio, providing Bigtable-like capabilities for Hadoop. That is, it provides a fault-tolerant way of storing large quantities of sparse data (small amounts of information caught within a large collection of empty or unimportant data). We use HBase version 0.94 here as our database for storing crawled webpages.

Elasticsearch is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java. We use Elasticsearch 1.4.1 for indexing our collected data from each search.

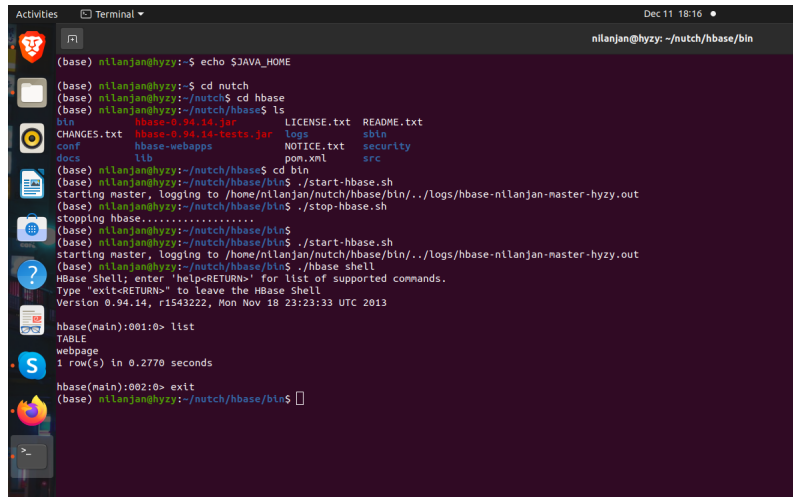
Our mini-project idea revolves around building a small scale crawler using Apache Nutch 2.3, building the database using Apache HBase 0.94 and indexing using Elasticsearch which can if needed, with enough parallel computing be expanded into a large scale web crawler. Note that we have used Ubuntu as our Operating system. This project can be done on Windows or any other operating system and the base steps will remain the same however the commands may vary. For any other Debian based OS, the commands will remain the same.

Working Steps

After downloading and extracting Apache Nutch, HBase and Elasticsearch in to a folder, we test the installation of HBase in the following figure (Fig.1) to check if it is installed correctly. Note that HBase requires `JAVA_HOME` to be set up. We can do that by just writing,

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

in the terminal. (Note that this is taking into consideration that we have java installed in our machine. If Java is not installed we can do that by the command `sudo apt install default-jre` and `sudo apt install default-jdk`. Usually Java is installed under `usr/lib`, however if the Java directory is something else, just set the path to `JAVA_HOME` path to that instead). In the `hbase-site.xml` file under the `hbase/conf` directory we make the necessary changes as to specify where our database will be stored.

A terminal window showing the installation and startup of HBase. The user is in a directory named 'nutch' and has navigated to 'hbase'. They run 'ls' and see files like 'CHANGELOG.txt', 'hbase-0.94.14-tests.jar', 'logs', 'sbin', 'conf', 'hbase-webapps', 'NOTICE.txt', 'security', 'docs', 'lib', 'pom.xml', and 'src'. They then run './start-hbase.sh', which starts the master and logs to a file. They run './stop-hbase.sh' to stop it. Then they run './start-hbase.sh' again. They enter the HBase shell and run 'list', which shows a table named 'webpage' with 1 row. Finally, they run 'exit' to leave the shell.

```
(base) nilanjan@hyzy:~$ echo $JAVA_HOME
(base) nilanjan@hyzy:~$ cd nutch
(base) nilanjan@hyzy:~/nutch$ cd hbase
(base) nilanjan@hyzy:~/nutch/hbase$ ls
CHANGELOG.txt  hbase-0.94.14-tests.jar  logs  sbin
conf           hbase-webapps            NOTICE.txt  security
docs          lib                      pom.xml     src
(base) nilanjan@hyzy:~/nutch/hbase$ ./start-hbase.sh
starting master, logging to /home/nilanjan/nutch/hbase/bin/../logs/hbase-nilanjan-master-hyzy.out
(base) nilanjan@hyzy:~/nutch/hbase/bin$ ./stop-hbase.sh
stopping hbase.....
(base) nilanjan@hyzy:~/nutch/hbase/bin$ ./start-hbase.sh
starting master, logging to /home/nilanjan/nutch/hbase/bin/../logs/hbase-nilanjan-master-hyzy.out
(base) nilanjan@hyzy:~/nutch/hbase/bin$ ./hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.94.14, r1543222, Mon Nov 18 23:23:33 UTC 2013
hbase(main):001:0> list
TABLE
webpage
1 row(s) in 0.2770 seconds
hbase(main):002:0> exit
(base) nilanjan@hyzy:~/nutch/hbase/bin$
```

Fig. 1: Checking if HBase is working properly

Simply running the command `hbase shell` after running the shell script `start-hbase.sh` confirms us that our HBase is properly installed and is running properly.

Now we open another terminal and run Elasticsearch, we can do it in the same terminal but it'll just look messy if we go back and forth on Elasticsearch and webcrawling. Simply executing the command `elasticsearch` in the bin folder of our Elasticsearch directory starts the Elasticsearch. Elasticsearch uses port 9200 of the localhost, so just writing `curl localhost:9200` confirms if Elasticsearch is running or not (Fig. 2). We can also check the browser to see if Elasticsearch is running or not (Fig. 3).

```
(base) milanjan@hyzy:~/nutch$ curl localhost:9200
{
  "status" : 200,
  "name" : "Meteorite",
  "cluster_name" : "elasticsearch",
  "version" : {
    "number" : "1.4.1",
    "build_hash" : "89d3241d670db65f994242c8e8383b169779e2d4",
    "build_timestamp" : "2014-11-26T15:49:29Z",
    "build_snapshot" : false,
    "lucene_version" : "4.10.2"
  },
  "tagline" : "You Know, for Search"
}
```

Fig. 2: Checking if Elasticsearch is working properly

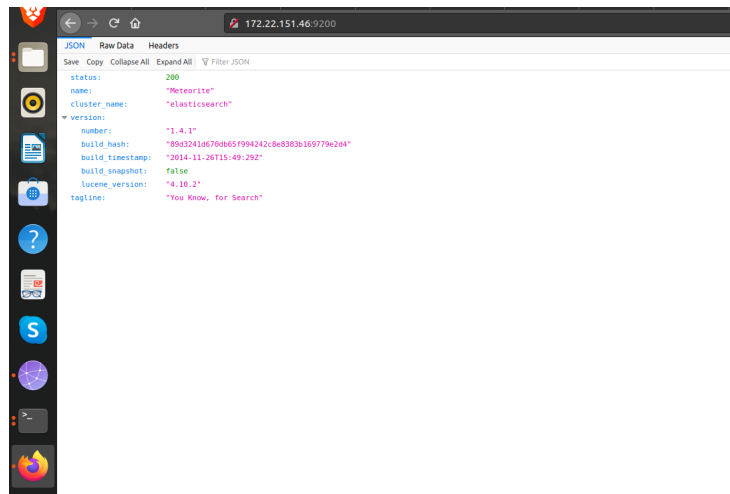


Fig. 3: Checking if Elasticsearch is working properly

After this, we set nutch up and with HBase as it's database in `nutch/conf/gora.properties`.

Next we run `ant runtime` inside the nutch folder. What it does is basically download all dependencies and puts them into a runtime folder.

To check if the runtime has been executed properly, we should have a runtime directory in nutch and `nutch/runtime/local/bin` there is supposed to be two shell scripts `crawl.sh` and `nutch.sh` which we will be using in a short time to run the crawl. (If Apache Ant has not been installed just type the command `sudo apt install ant` which will download the required packages and install the software).

The next step is to set some properties required for the crawl in `nutch/runtime/local/conf/nutch-site.xml` like naming our crawler because when we crawl a website, and it sees that we have no agent name, it may block our IP since it may think that it is some kind of a DDOS attack.

After that we set HBase as our storage. The final property needed for a basic crawl, we indicate the basic plugins (including the Elasticsearch dependencies) needed for crawling. (Fig. 2)

Now there are a lot of properties that we can use, like we can limit if a page sends us pdfs which will be downloaded, we can continue crawling a website even if it does not have a robots.txt file in their root, we can use a proxy so that in case if we get banned, our computer does not get banned and we can just use a new proxy. Details about all the properties we can set can be found at `nutch/runtime/local/conf/nutch-default.xml`.

```

1 <?xml version="1.0"?>
2 <!DOCTYPE nutch SYSTEM "nutch.xml" >
3 <configuration>
4 <!-- The site specific properties override to this file -->
5
6 <property>
7   <name>nutch.agent.name</name>
8   <value>NutchCrawler</value>
9 </property>
10
11 <property>
12   <name>nutch.data.store.class</name>
13   <value>org.apache.hadoop.hbase.store.HBaseStore</value>
14 </property>
15
16 <property>
17   <name>nutch.plugins.includes</name>
18   <value>protocol-http|urlfilter-regex|parser-(html|tika)|index-(basic|anchor)|urlnormalizer-(pass|regex|basic)|scoring-opic</value>
19 </property>
20
21 <property>
22   <name>nutch.elastic.host</name>
23   <value>localhost</value>
24 </property>
25
26 <property>
27   <name>nutch.elastic.port</name>
28   <value>9300</value>
29 </property>
30
31 <property>
32   <name>nutch.elastic.cluster</name>
33   <value>elasticsearch</value>
34 </property>
35
36 <property>
37   <name>nutch.elastic.index</name>
38   <value>nutch</value>
39 </property>
40
41 <property>
42   <name>nutch.elastic.index.name</name>
43   <value>nutch</value>
44 </property>
45
46 <property>
47   <name>nutch.elastic.index</name>
48   <value>nutch</value>
49 </property>
50
51 </configuration>

```

Fig. 4: Our nutch-site.xml file where we name our crawler and add its dependencies and plugins

Next we need to seed nutch i.e. we need to tell nutch from which websites we want to start crawling. For that we need to make a new folder in `nutch/runtime/local` called `urls`. In that folder we need to make a seed.txt where we'll have the sites we want as seed (Fig.5).

```

1 https://apple.com

```

Fig. 5: Our seed.txt file

We just seed with one url because we are using a single threaded machine and it will take a lot of time to seed from multiple urls (as we will see later). Here

we take our seed url as <https://apple.com>.

Next in the `regex-urlfilter.txt` we set the regex which we want our web crawler to crawl. This is very necessary for a small scale crawler due to the same reason as above. It will take days in our single threaded machine and our crawl will branch out which for the sake of time complexity we don't want to happen.

Finally now we can start the crawl.

The first step in the crawl algorithm is we have to seed nutch i.e. we have to tell it where to start crawling..

```
(base) n1lanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch inject urls
InjectorJob: starting at 2020-12-07 13:57:06
InjectorJob: Injecting urlDir: urls
InjectorJob: Using class org.apache.gora.hbase.store.HBaseStore as the Gora storage class.
InjectorJob: total number of urls rejected by filters: 0
InjectorJob: total number of urls injected after normalization and filtering: 1
::: Injector: finished at 2020-12-07 13:57:09, elapsed: 00:00:02
::: (base) n1lanjan@hyzy:~/nutch/nutch/runtime/local$
```

Fig. 6: Injecting the seed urls to the crawl

As shown in the figure (Fig. 6), we do it by using the command `nutch inject urls`. What it does is nutch will search for a folder named `urls` and will search for a `seed.txt` inside that folder which contains the seed urls. The output shows the number of URLs that has been rejected after the regex filter, the number of urls accepted and the time required for doing so.

Next we set the max number of urls that we would like to fetch for the upcoming crawl. We use the command `nutch generate -topN 1000` for that purpose. Note that here we set the maximum number of urls to be fetched as 1000. We can select this to be any number however due to time complexity constraints, we are choosing a low number. A bigger web crawler will usually have no such limit set. Here the top N urls are ranked on the basis of their popularity (Fig. 7).

```
(base) n1lanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch inject urls
InjectorJob: starting at 2020-12-07 13:57:06
InjectorJob: Injecting urlDir: urls
InjectorJob: Using class org.apache.gora.hbase.store.HBaseStore as the Gora storage class.
InjectorJob: total number of urls rejected by filters: 0
InjectorJob: total number of urls injected after normalization and filtering: 1
::: Injector: finished at 2020-12-07 13:57:09, elapsed: 00:00:02
::: (base) n1lanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch generate -topN 1000
GeneratorJob: starting at 2020-12-07 13:58:31
GeneratorJob: Selecting best-scoring urls due for fetch.
GeneratorJob: starting
GeneratorJob: filtering: true
GeneratorJob: normalizing: true
GeneratorJob: topN: 1000
GeneratorJob: finished at 2020-12-07 13:58:33, time elapsed: 00:00:01
::: GeneratorJob: generated batch id: 1007329711-1111397273 containing 1 URLs
::: (base) n1lanjan@hyzy:~/nutch/nutch/runtime/local$
```

Fig. 7: Setting the number of max urls to be fetched

As we can see that the generated batch contains 1 url which is our seed url. The next step is to fetch the url that we have generated i.e. `https://apple.com`.

For that we use the command `nutch fetch -all` as shown in Fig. 8.

```
(base) n1lanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch inject urls
InjectorJob: starting at 2020-12-07 13:57:06
InjectorJob: Injecting urlDir: urls
InjectorJob: Using class org.apache.gora.hbase.store.HBaseStore as the Gora storage class.
InjectorJob: total number of urls rejected by filters: 0
InjectorJob: total number of urls injected after normalization and filtering: 1
Injector: finished at 2020-12-07 13:57:09, elapsed: 00:00:02
(base) n1lanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch generate -topN 1000
GeneratorJob: starting at 2020-12-07 13:58:31
GeneratorJob: Selecting best-scoring urls due for fetch.
GeneratorJob: starting
GeneratorJob: filtering: true
GeneratorJob: normalizing: true
GeneratorJob: topN: 1000
GeneratorJob: finished at 2020-12-07 13:58:33, time elapsed: 00:00:01
GeneratorJob: generated batch id: 1607329711-1111397273 containing 1 URLs
(base) n1lanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch fetch -all
FetcherJob: starting at 2020-12-07 14:00:48
FetcherJob: fetching all
FetcherJob: threads: 10
FetcherJob: parsing: false
FetcherJob: resuming: false
FetcherJob: timelimit set for : -1
Using queue mode : byHost
Fetcher: threads: 10
QueueFeeder finished: total 1 records. Hit by time limit :0
fetching https://apple.com/ (queue crawl delay=5000ms)
-finishling thread FetcherThread2, activeThreads=1
-finishling thread FetcherThread1, activeThreads=1
-finishling thread FetcherThread3, activeThreads=1
-finishling thread FetcherThread4, activeThreads=1
-finishling thread FetcherThread5, activeThreads=1
-finishling thread FetcherThread6, activeThreads=1
-finishling thread FetcherThread7, activeThreads=1
-finishling thread FetcherThread8, activeThreads=1
Fetcher: throughput threshold: -1
-finishling thread FetcherThread9, activeThreads=1
Fetcher: throughput threshold sequence: 5
-finishling thread FetcherThread0, activeThreads=0
0/0 spinwaiting/active, 1 pages, 0 errors, 0.2 0 pages/s, 0 0 kb/s, 0 URLs in 0 queues
-activeThreads=0
FetcherJob: finished at 2020-12-07 14:00:55, time elapsed: 00:00:07
(base) n1lanjan@hyzy:~/nutch/nutch/runtime/local$
```

Fig. 8: Fetching the urls we generated in the previous step

As we can see that we have 1 page `https\\apple.com` with 0 errors.

The next step is to parse the urls that we just fetched (Fig. 9). What this step does is Nutch will go through all the urls we just fetched and pull out the data and any other urls for the next fetch.

To do that we use the command `nutch parse -all`.

```
(base) n1lanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch parse -all
ParserJob: starting at 2020-12-07 14:01:24
ParserJob: resuming: false
ParserJob: forced reparse: false
ParserJob: parsing all
Parsing https://apple.com/
ParserJob: success
ParserJob: finished at 2020-12-07 14:01:27, time elapsed: 00:00:02
(base) n1lanjan@hyzy:~/nutch/nutch/runtime/local$
```

Fig. 9: Parsing the urls fetched in the previous step

The last step is to update the database command. `nutch updatedb -all` which updates our HBase database with the results of the latest crawl (Fig. 10).

```
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch updatedb -all
DbUpdaterJob: starting at 2020-12-07 14:01:43
DbUpdaterJob: updating all
DbUpdaterJob: finished at 2020-12-07 14:01:45, time elapsed: 00:00:02
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$
```

Fig. 10: Updating the HBase database

The next step is to index the results that we have gotten. Till now we have not yet begun the actual crawl, these are just the preliminary steps required to go through before the crawl, so we will index after the first crawl. Also an interesting thing we noticed is that if we seed the url `https://apple.com`, we are redirected to `https://www.apple.com`. So if we run the fetch a second time after this, we get only one page which is `https://www.apple.com`. What we do

```
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch generate -topN 10000
GeneratorJob: starting at 2020-12-07 14:42:40
GeneratorJob: Selecting best-scoring urls due for fetch.
GeneratorJob: starting
GeneratorJob: filtering: true
GeneratorJob: normalizing: true
GeneratorJob: topN: 10000
GeneratorJob: finished at 2020-12-07 14:42:41, time elapsed: 00:00:01
GeneratorJob: generated batch id: 1607332360-14480712 containing 1 URLs
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch fetch -all
FetcherJob: starting at 2020-12-07 14:43:10
FetcherJob: fetching all
FetcherJob: threads: 10
FetcherJob: parsing: false
FetcherJob: resuming: false
FetcherJob: timeout set for : -1
Using queue mode : byHost
Fetcher: threads: 10
QueueFeeder finished: total 1 records. Hit by time limit: 0
-finish thread FetcherThread0, activeThreads=1
-finish thread FetcherThread4, activeThreads=1
-finish thread FetcherThread2, activeThreads=1
-finish thread FetcherThread3, activeThreads=1
fetching https://www.apple.com/ (queue crawl delay=5000ms)
-finish thread FetcherThread5, activeThreads=1
-finish thread FetcherThread6, activeThreads=1
-finish thread FetcherThread7, activeThreads=1
-finish thread FetcherThread8, activeThreads=1
Fetcher: throughput threshold: -1
Fetcher: throughput threshold sequence: 5
-finish thread FetcherThread9, activeThreads=1
-finish thread FetcherThread1, activeThreads=0
0/0 spmWaiting/active, 1 pages, 0 errors, 0.2 0 pages/s, 0 URLs in 0 queues
-activeThreads=0
FetcherJob: finished at 2020-12-07 14:43:18, time elapsed: 00:00:07
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch parse -all
ParserJob: starting at 2020-12-07 14:43:49
ParserJob: resuming: false
ParserJob: forced reparse: false
ParserJob: parsing all
ParserJob: success
ParserJob: finished at 2020-12-07 14:43:51, time elapsed: 00:00:02
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch updatedb -all
DbUpdaterJob: starting at 2020-12-07 14:44:17
DbUpdaterJob: updating all
DbUpdaterJob: finished at 2020-12-07 14:44:19, time elapsed: 00:00:01
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$
```

Fig. 11: Fetching `www.apple.com` in the second run of the crawler

now is repeat the same steps as above so that our HBase database is updated with `https://www.apple.com` as shown in Fig. 11. After we do that, we are ready to start fetching new urls from `https://www.apple.com`.

```
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch generate -topN 1000
GeneratorJob: starting at 2020-12-07 14:44:50
GeneratorJob: Selecting best-scoring urls due for fetch.
GeneratorJob: starting
GeneratorJob: filtering: true
GeneratorJob: normalizing: true
GeneratorJob: topN: 1000
GeneratorJob: finished at 2020-12-07 14:44:52, time elapsed: 00:00:01
GeneratorJob: generated batch id: 1607332490-2020014223 containing 99 URLs
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$
```

Fig. 12: Generating the top 1000 urls

Now if we generate the top 1000 urls, since we have the parsed `https://www.apple.com` in our database, we should get all the links from that page.

As we can see, we have generated 99 urls which are all the urls (Fig. 12) that can be found from the site `https://www.apple.com`.

```
Activities Terminal ▾ Dec 7 14:47 • nilanjan@hyzy: ~/nutch/nutch/runtime/local
GeneratorJob: topN: 1000
GeneratorJob: finished at 2020-12-07 14:44:52, time elapsed: 00:00:01
GeneratorJob: generated batch id: 1607332490-2020014223 containing 99 URLs
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch fetch -all
FetcherJob: starting at 2020-12-07 14:45:28
FetcherJob: fetching all
FetcherJob: threads: 10
FetcherJob: parsing: false
FetcherJob: resuming: false
FetcherJob: timelimit set for : -1
Using queue mode : byHost
Fetcher: threads: 10
fetching https://www.apple.com/it/ (queue crawl delay=5000ms)
Fetcher: throughput threshold: -1
Fetcher: throughput threshold sequence: 5
QueueFeeder finished: total 99 records. Hlt by time limit :0
fetching https://www.apple.com.cn/ (queue crawl delay=5000ms)
10/10 spinning/waiting/active, 2 pages, 0 errors, 0.4 0 pages/s, 204 204 kb/s, 97 URLs in 1 queues
fetching https://www.apple.com/au/ (queue crawl delay=5000ms)
10/10 spinning/waiting/active, 3 pages, 0 errors, 0.3 0 pages/s, 153 102 kb/s, 96 URLs in 1 queues
fetching https://www.apple.com/kw/ (queue crawl delay=5000ms)
10/10 spinning/waiting/active, 4 pages, 0 errors, 0.3 0 pages/s, 136 102 kb/s, 95 URLs in 1 queues
fetching https://www.apple.com/kw-ar/ (queue crawl delay=5000ms)
10/10 spinning/waiting/active, 5 pages, 0 errors, 0.3 0 pages/s, 128 102 kb/s, 94 URLs in 1 queues
fetching https://www.apple.com/ne/ (queue crawl delay=5000ms)
10/10 spinning/waiting/active, 6 pages, 0 errors, 0.2 0 pages/s, 116 72 kb/s, 93 URLs in 1 queues
```

Fig. 13: Fetching the webpages

Following the procedure as explained before, we fetch the urls and parse them and finally update our HBase database with our newly crawled data.

As we can see from the next figure (Fig. 14) that just a simple fetch from one single website (which are filtered by regex) takes over 9 minutes in a single threaded machine. Now it comes back to our previous idea of seeding just one url. Imagine what would have happened if we seeded something like 100 urls! It would take days!


```

0. https://www.apple.com/hu/
1. https://www.apple.com/hu/
fetching https://www.apple.com/hu/ (queue crawl delay=5000ms)
0/10 spinning/active, 98 pages, 0 errors, 0.2 0 pages/s, 77 98 kb/s, 1 URLs in 1 queues
+ queue: https://www.apple.com
+ nextFetchTime = 1
+ inProgress = 0
+ crawlDelay = 5000
+ minCrawlDelay = 0
+ nextFetchTime = 100733088088
now = 100733085478
0. https://www.apple.com/hu/
fetching https://www.apple.com/hu/ (queue crawl delay=5000ms)
-finish thread FetcherThread5, activeThreads=0
-finish thread FetcherThread7, activeThreads=2
-finish thread FetcherThread9, activeThreads=4
-finish thread FetcherThread9, activeThreads=3
-finish thread FetcherThread9, activeThreads=0
-finish thread FetcherThread3, activeThreads=7
-finish thread FetcherThread1, activeThreads=1
-finish thread FetcherThread2, activeThreads=0
0/0 spinning/active, 99 pages, 0 errors, 0.2 0 pages/s, 77 101 kb/s, 0 URLs in 0 queues
+ activeThreads=0
FetcherJob: finished at 2020-12-07 14:54:51, time elapsed: 00:09:22
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$

```

Fig. 14: Fetching the webpages

The next step after completing the parsing (Fig. 15)(which also shows us all the webpages that have been fetched in this crawl) and updating the database (Fig. 17) to HBase, is that we are going to index the database with the help of Elasticsearch 1.4.1 so that we can actually use the data we just collected for searching or anything else.

```

Activities Terminal Dec 7 17:37
nilanjan@hyzy: ~/nutch/nutch/runtime/local
0/0 spinning/active, 99 pages, 0 errors, 0.2 0 pages/s, 77 101 kb/s, 0 URLs in 0 queues
-activeThreads=0
FetcherJob: finished at 2020-12-07 14:54:51, time elapsed: 00:09:22
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch parse -all
ParserJob: starting at 2020-12-07 17:37:26
ParserJob: resuming: false
ParserJob: forced reparse: false
ParserJob: parsing all
Parsing https://www.apple.com/cn/
Parsing https://www.apple.com/ae-ar/
Parsing https://www.apple.com/ae/
Parsing https://www.apple.com/an/
Parsing https://www.apple.com/at/

```

Fig. 15: Parsing the webpages

```

Parsing https://www.apple.com/tw/
Parsing https://www.apple.com/ug/
Parsing https://www.apple.com/uk/
Parsing https://www.apple.com/us/shop/goto/bag
https://www.apple.com/us/shop/goto/bag skipped. Content of size 20 was truncated to 0
Parsing https://www.apple.com/vn/
Parsing https://www.apple.com/za/
ParserJob: success
ParserJob: finished at 2020-12-07 17:37:30, time elapsed: 00:00:04
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$

```

Fig. 16: Parsing the webpages

```

(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$ bin/nutch updatedb -all
DbUpdaterJob: starting at 2020-12-07 17:38:05
DbUpdaterJob: updating all
DbUpdaterJob: finished at 2020-12-07 17:38:08, time elapsed: 00:00:02
(base) nilanjan@hyzy:~/nutch/nutch/runtime/local$

```

Fig. 17: Updating the database

Now, our last task is to run `nutch index -all` command which indexes all the data via the help of Elasticsearch (Fig. 18).

```
(base) ntlanjanghyzyi:~/nutch/nutch/runtime/local$ bin/nutch index -all
IndexingJob: starting
Active IndexWriters :
ElasticIndexWriter
  elastic.cluster : elastic prefix cluster
  elastic.host : hostname
  elastic.port : port (default 9300)
  elastic.index : elastic index command
  elastic.max.bulk.docs : elastic bulk index doc counts. (default 250)
  elastic.max.bulk.size : elastic bulk index length. (default 2500500 ~2.5MB)

IndexingJob: done.
(base) ntlanjanghyzyi:~/nutch/nutch/runtime/local$
```

Fig. 18: Indexing the data

To check if we have correctly indexed all the documents, we can just check the number of indices from the port 9200 (Fig. 19).



health status index pri rep docs.count docs.deleted store.size pri.store.size
yellow open nutch 5 1 95 0 480.8kb 480.8kb

Fig. 19: Checking if all the data is indexed

Finally our crawling and indexing is done. Now we can submit a request to the REST API server and we will be given a json version of the data that we ask for. Here we ask for all the data and we are given all the data from the database in json format by using the command

```
curl -l -H "Accept:application/json" -H "Content-Type: application/json"
http://172.22.151.46:9200/nutch/_search as we can see in Fig. 20.
```

[illegible]

Fig. 20: Submitting a request to the rest api server

We can convert that into readable json format and that will give us a general idea about what kind of crawling nutch does. We just take a snippet of the massive json output that we recieved, and run it through a json formatter to get a readable output where we can easily read through the contents of the webpages that we fetched (Fig.21).

```
{
  "took": 35,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 95,
    "max_score": 1.0,
    "hits": [
      {
        "_index": "nutch",
        "_type": "doc",
        "_id": "cn.com.apple.www:https/",
        "_score": 1.0,
        "_source": {
          "timestamp": "2021-01-06T09:15:32.148Z",
          "digest": "d64e6248488fcb9591398edd2e2d883",
          "host": "www.apple.com.cn",
          "boost": "0.0",
          "id": "cn.com.apple.www:https/",
          "title": "Apple (中国大陆) - 官方网站",
          "url": "https://www.apple.com.cn/",
          "content": "Apple (中国大陆) - 官方网站 <广告> Global Nav 打开菜单 Global Nav 关闭菜单 Apple 购物袋 + 搜索 apple.com.cn 取消 Apple Mac iPad iPhone Watch Music 技术支持 购物袋 + 取消 登录/注册, ..."
        }
      },
      {
        "_index": "nutch",
        "_type": "doc",
        "_id": "com.apple.www:https/ca/",
        "_score": 1.0,
        "_source": {
          "timestamp": "2021-01-06T09:22:18.449Z",
          "digest": "4942c33217045daf6b6c3097f4323341",
          "host": "www.apple.com",
          "boost": "0.0",
          "id": "com.apple.www:https/ca/",
          "title": "Apple (Canada)",
          "url": "https://www.apple.com/ca/",
          "content": "Apple (Canada) Apple Global Nav Open Menu Global Nav Close Menu Apple Shopping Bag + Search apple.com/ca Cancel Apple Mac iPad iPhone Watch TV Music Support Shopping Bag + Cancel Shop early onl ..."
        }
      }
    ]
  }
}
```

Fig. 21: A snippet of the output that we got, viewed nicely with json formatter

Conclusion

Apache Nutch, Apache HBase and Elasticsearch are some of the most used softwares in the corporate world. In this project we learnt how to make a small scale web crawler, crawl a small section of the web and properly store and index the data that we crawled using Apache Nutch, HBase and Elasticsearch.

References

- [1]Apache Nutch official tutorial
- [2]Apache Nutch 2.0 with Elastisearch
- [3]Apache HBase
- [4]Elastisearch 1.4.1