山-shan
Scalable Search and Web Crawling

Carlos López Roa
me@mr3m.me
DMKM-UPO

February 10, 2017

Abstract

The objective of this work was to take the concepts of information retrieval to implement a scalable framework for the general task of indexing unstructured documents and retrieve them from the web. Our case study was to take Wikipedia data as crawlable and indexable target. After crawling and indexing, a GUI, deployed in the cloud, displays the results and allows the user to do personalised queries. Shan (山) is the chinese character for mountain. It can also be composed concatenating the first letter of the components: Solr Hadoop Apache Nutch. The code for the project it's available in the repository github.com/mr3m/SHAN. For a live demo please visit mr3m.me/shan

1 Introduction

In the field of Information Retrieval (IR), the task of full-text search is often approached by constructing an index, so that the user can query the body of knowledge and obtain a ranked list of documents that match his query.

Also there exists interest for automating the information gathering in the World Wide Web using automatic distributed query programs (bots or spiders) for crawling the massive corpus of information contained in the web, in parallel but reducing redundancy working collaboratively.

In this work we took the approach to implement a general framework that could:

1. index unstructured documents
2. crawl in the world wide web unstructured documents
3. correctly answer custom queries
4. scale based on a distributed architecture

Hence, our choice of frameworks to correctly address was to use the Apache family of technologies from the Apache Software Foundation. So, for each goal we can choose a diferente framework and they can easily interact.

1. Apache Lucene for scalable high performance indexing
2. Apache Nutch for highly scalable web crawling
3. Apache Solr for blazing-fast search platform
4. Docker for building shipping and running the different virtual instances

We chose to take the Wikipedia body of knowledge as a case study to demonstrate the capabilities of the framework solution to scale to GB scale data source.

2 Implementation

The proposed stack consists in a container based micro service structure, where instead of a monolithic application in a single stack, a swarm of services will provide independent functions. This helps to provide scalability and resilience. Hence, a multi instance container based in docker will serve as common operating system infrastructure.

- The indexing engines will be served in independent containers running Solr
- And web crawling tasks will be done by another group of containers running Nutch

A sketch of the architecture can be seen in figure 1
The development was done in a personal laptop and the deployment was done in Cloud infrastructure in Amazon Web Services. For a live demo please visit mr3m.me/shan

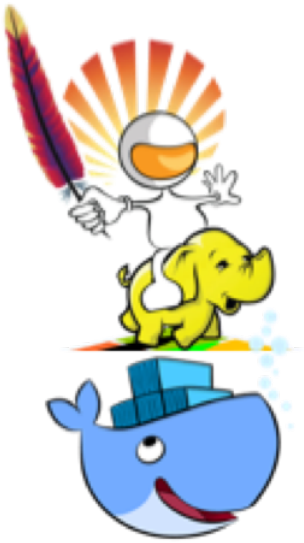

Figure 1: A general overview of the architecture. We have Apache Web server on top of Solr, with Luscene in the background. This is on top of Hadoop. The whole stack resides inside docker containers.

2.1 WebCrawling

Apache Nutch is a well matured, production ready Web crawler. Nutch provides extensible interfaces such as Parse, Index and ScoringFilter's for custom implementations. We can find Web page hyperlinks in an automated manner, reduce lots of maintenance work, for example checking broken links, and create a copy of all the visited pages for searching over.

The procedure taken was:

1. Set the Crawler properties, such as name, delay time and number of threads
2. Restrict the crawler to stay in our preferred domain, `wikipedia.org`
3. Inject a seed file with the URLs from which we want to begin crawling, in this case `en.wikipedia.org`
4. Generate a crawling db from the seed by fetching it, capturing the links inside and parsing them.
5. Fetch those URLs, download the content
6. Parse the fetched content
7. Repeat until the desired results are achieved.

This workflow was implemented using a Docker container of the Apache/nutch repository.

2.2 Indexing

Apache Solr is an open source search platform built upon a Java library called Lucene. Solr is a popular search platform for Web sites because it can index and search multiple sites and return recommendations for related content based on the search query's taxonomy.

The steps followed were:

1. Create and initialise a Solr core, with a custom made schema.xml file to handle the desired fields
2. Given the crawled data create a RequestHandler that can parse the data using the internal capabilities of the library for reading and parsing XML and generating structured features from the unstructured data set.
3. Call the RequestHandler on the data and wait for it to finish
4. Populate the index in the core
5. Initialise the web application and query.

The output of the RequestHandler REST API can be found in the figure 2.2


```
▼<response>
  ▼<lst name="responseHeader">
     <int name="status">0</int>
     <int name="QTime">5</int>
   </lst>
  ▼<lst name="initArgs">
    ▼<lst name="defaults">
       <str name="config">data-config.xml</str>
     </lst>
   </lst>
   <str name="status">idle</str>
   <str name="importResponse"/>
  ▼<lst name="statusMessages">
     <str name="Total Requests made to
     DataSource">0</str>
     <str name="Total Rows Fetched">17098961</str>
     <str name="Total Documents
     Processed">11505840</str>
     <str name="Total Documents
     Skipped">5593121</str>
     <str name="Full Dump Started">2017-02-08
     06:24:43</str>
    ▼<str name="">
       Indexing completed. Added/Updated: 11505840
       documents. Deleted 0 documents.
     </str>
     <str name="Committed">2017-02-08
     10:59:37</str>
     <str name="Time taken">2:17:23.339</str>
   </lst>
 </response>
```


Figure 2: A total of 17,098,961 documents found and processed 11,505,840. processed in 137 minutes

This workflow was implemented in a Docker container of the Apache/Solr repository.

2.3 Cloud Deployment

Using the technology of Amazon Elastic Cloud Computing (EC2), that allow us to spin virtual machines in the cloud and the Elastic Container Registry

(ECR) that allow us to push and pull docker images in a private and secure repository, we simply, committed, pushed and then pulled the generated Docker image, following a dev-ops approach. That is, all the development environment was quickly put into production environment by just committing, pushing and pulling as one does in Git related version control software, but in this case the version control it's done in the virtual machine guest, it's non-volatile state and storage.

A screenshot of the resulting interface can be seen in figure **??**

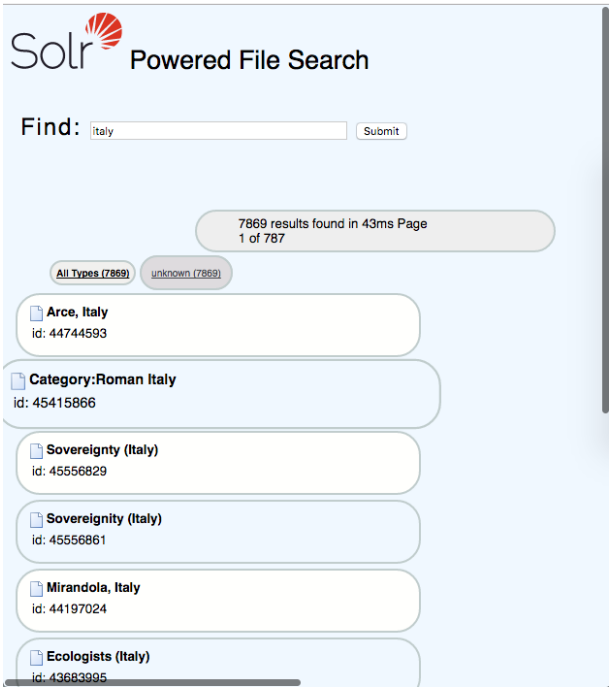

Figure 3: Implementd interface running in cloud infrastructure. Searching for query `italy` amongst 11,505,840 documents. 7,869 results found in 43ms. labelfig3

3 Conclusions and Future work

- We were able to implement a solution meeting the goals proposed
- Using the Apache family of solutions resulted in a fast and agile development, providing core solutions fast
- Web crawling it's a very distributable, here we worked in single core, 10 threads, but the framework it's ready for
- Indexing it's extremely fast, processing around 40GB of data took around 120 minutes
- Also search it's lightning fast, selecting 0.06% of the data takes only half 0.043 seconds
- Some customisations in the interface and functionalities are possible, but they are domain dependant.
- The implemented framework has already been implemented by the author in another domain, indexing diverse data sources (word, powerpoint files, emails, text files, images) and providing the search interface for them.

References

[1] Web Crawling and Data Mining with Apache Nutch - Laliwala, Zakir, Shaikh, Abdulbasit Fazalm