CS242 Project – Part B submission

LOTR INFORMATION RETRIEVAL

JORGE MERCADO, hOVANES KESEYAN, Janakiram Kuppa

2018

Contents

[1. Objective 2](#_Toc508454872)

[2. Collaboration details: 2](#_Toc508454873)

[2.1. Team Tasks 2](#_Toc508454874)

[2.2. Jorge Mercado 2](#_Toc508454875)

[2.3. Hovanes Keseyan: 3](#_Toc508454876)

[2.4. Janakiram Kuppa: 3](#_Toc508454877)

[3. System Overview: 4](#_Toc508454878)

[3.1. Overall System Architecture 4](#_Toc508454879)

[3.2. Updated Crawling Architecture 6](#_Toc508454880)

[4. Modified Lucene Indexing Architecture 7](#_Toc508454881)

[4.1. Modified Indexing Architecture 7](#_Toc508454882)

[5. Map Reduce Inverted Index Architecture and process 8](#_Toc508454883)

[5.1. Map Reduce Inverted Index Architecture 8](#_Toc508454884)

[5.2. How was Hadoop used 9](#_Toc508454885)

[5.3. Explanation of Indexes built by Hadoop 9](#_Toc508454886)

[5.4. Explanation of how Hadoop-created inverted index to do ranking 13](#_Toc508454887)

[5.5. Explanation of how Lucene index was used to return results 13](#_Toc508454888)

[6. Limitations of the system 14](#_Toc508454889)

[7. Obstacles and solutions 14](#_Toc508454890)

[8. Index Process time comparison between Lucene and Map Reduce Inverted Index 14](#_Toc508454891)

[9. How to deploy the system 14](#_Toc508454892)

[10. Screenshots of the system in action 15](#_Toc508454893)

# Objective

Objective of this project is to a create web crawler to crawl websites that contain data about characters, their details, locations in the world, and other information gathered from plot points and stories told in the world created in legendary author J.R.R. Tolkien’s works: The Lord of the Rings, The Hobbit, and The Silmarillion. Intent is to crawl a large and popular online Wiki located at http://lotr.wikia.com, known as “The One Wiki to Rule Them All”. This wiki is populated by fans with data from all the literature behind this world, including not only the original books but other canonical media such as the popular and successful movie series from the 2000s. Each character has a wiki page which describes any known information about their timeline, city/town of origin, actors that have portrayed them in media, relationships to other characters, and important plot points among the immense library of information that is available.

The data collected from the crawler is to be indexed using Lucene first, and later using Hadoop. Create a web page to search both Lucene and Inverted Indexes for the keywords entered in the query, and show the webpages that meet the query criteria and show the top results.

# Collaboration details:

## Team Tasks

All the team members worked together as a team on the following tasks:

1. Finalize JSON doc schemas (team)
2. System Test search engine with all functionality in place
3. Record Presentation Video
4. Finalize Report

## Jorge Mercado

Jorge worked on the following tasks during this phase of the project:

* 1. Clean up Crawler code from prior phase of the project and generate 5 GB worth of data
  2. Come up with an approach to merge 35K XML files to be just 50-60 files
  3. Create content JSON data from XML files and load it into Mongo DB
  4. Create python code to search using Lucene index (java REST wrapper)
  5. Create python code to search using inverted index from MongoDB (worked with Hovanes on this task)
  6. Create Lucene index for 5 GB data
  7. Create web app interface for search
  8. Unit test Search engine for Lucene index and make sure it works
  9. Unit test Search engine for inverted index from MR and make sure it works
  10. Incorporate snippets into search

## Hovanes Keseyan:

Jorge worked on the following tasks during this phase of the project:

1. Generate sample JSON file for content and import into MongoDB
2. Generate sample JSON file for inverted index and import into MongoDB
3. Review the JSON formats for snippet generation, map reduce inverted index generation and make adjustments needed
4. Load inverted index JSON data into Mongo DB
5. Chart MapReduce performance graph
6. Search engine logic diagram for document/presentation
7. Create python code to search using inverted index from MongoDB (worked with Jorge on this task)
8. Create python code to generate snippets from MongoDB (worked with Jorge on this task)
9. Review and update some sections of the report document
10. Create Presentation PowerPoint (worked with Hovanes on this task)

## Janakiram Kuppa:

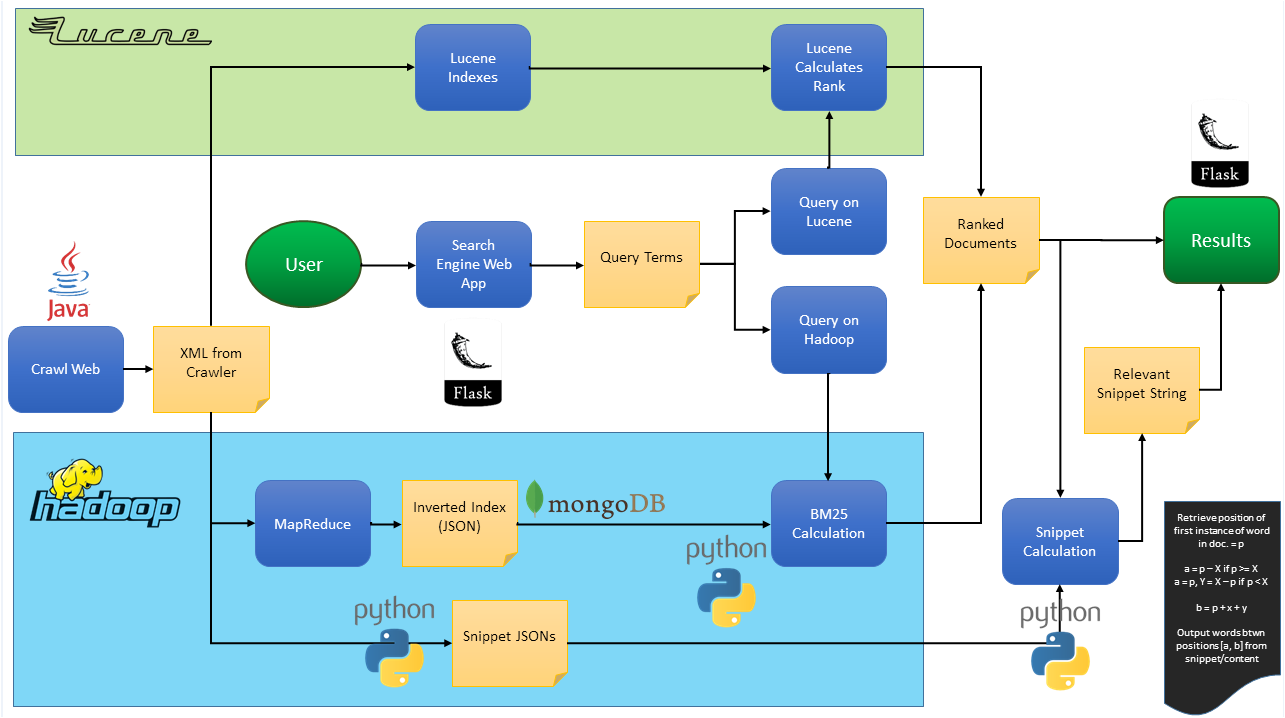
Janakiram worked on the following tasks during this phase of the project:

1. Coordinated the team effort on the project, requested/organized meetings, identified action items, assigned due dates and monitored progress.
2. Proposed the overall design to use HBase to store crawler data for snippet generation, store inverted index data in a HBase table as well. After getting clarification on this from TA, team decided to move towards MongoDB.
3. Designed the JSON formats for snippet generation, map reduce inverted index generation
4. Code the entire MapReduce inverted index logic to create JSON output with Inverted Index data
5. Unit test in Hadoop using 3 XML files
6. Create draft documentation for all aspects of the project
7. Load XML files related to 5 GB worth of data to Hadoop
8. Run inverted index map reduce on Hadoop to generate final set of JSON files
9. System test search engine
10. Create Presentation PowerPoint (worked with Hovanes on this task)

# System Overview:

## Overall System Architecture

**Figure 1: Overall System Architecture**



Overall System architecture comprises of using Java to crawl the web with the seed page of <http://lotr.wikia.com> and generate 6 GB worth of XML data output. XML data output is the input to the following streams:

1. Lucene Index generation stream that takes the XML data as input and creates Lucene index files
2. Map Reduce based inverted index process that takes the XML data as input and creates JSON Inverted Index files.

Lucene index algorithm uses BM25 calculation to do the ranking. For Inverted index output, python code was developed to do BM25 calculation so that the ranking is comparable between Lucene and Map Reduce based inverted index process.

There are 2 types of JSON outputs that are persisted in Mongo DB:

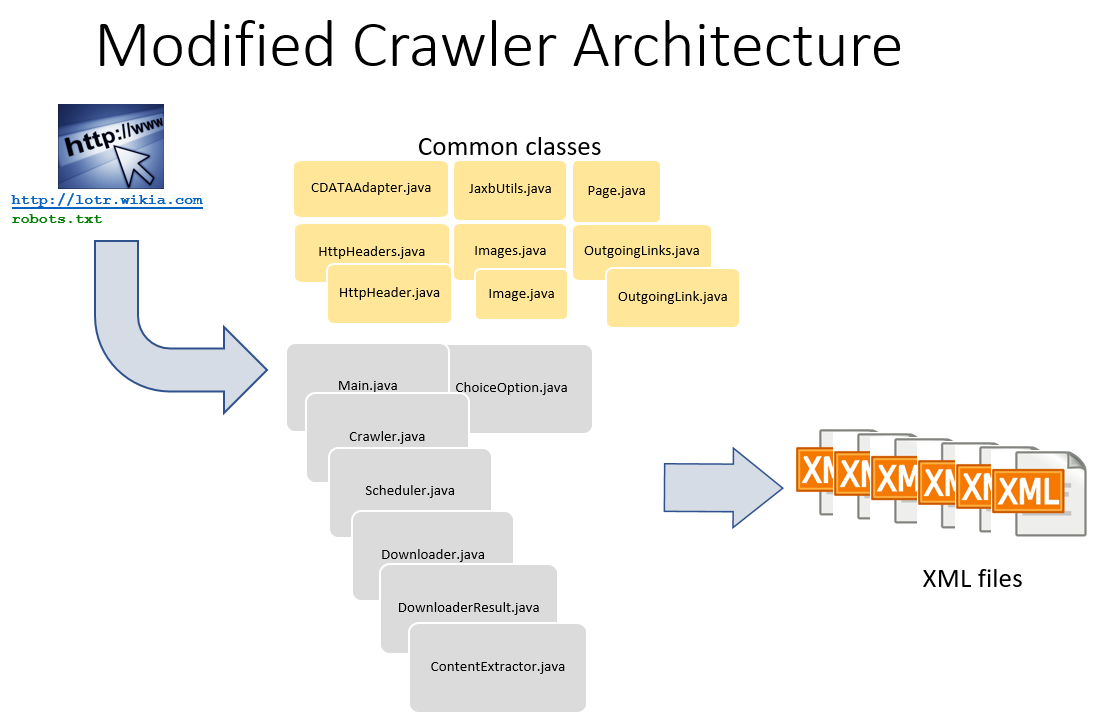
1. Inverted Index JSON (output from the Map Reduce process)
2. Content JSON (output from the crawler to persist the title and content portions of the generated XML)

Web search engine was developed using Flask/Python that will take the user search query, look up either Lucene Index output or the Inverted Index MongoDB collection that holds the Map Reduce inverted index output. Then the results from the search are ordered based on BM25 values. Snippet generation code was developed in Python to look up the “title” and “content” portions of the XML from the Content JSON stored in MongoDB. Snippets are generated using a sliding logic that zeroes in on the positions within the title and content portions of the content JSON and gets about 25 words that are within the range of the position where the terms of the query exist within the document.

## Updated Crawling Architecture

Crawler Architecture was updated to capture the outgoing links from a given web page. Also, the crawler code was optimized to process and provide 5 GB worth of data as XML output from the crawling process execution.

**Figure 2: Modified Crawler Architecture**

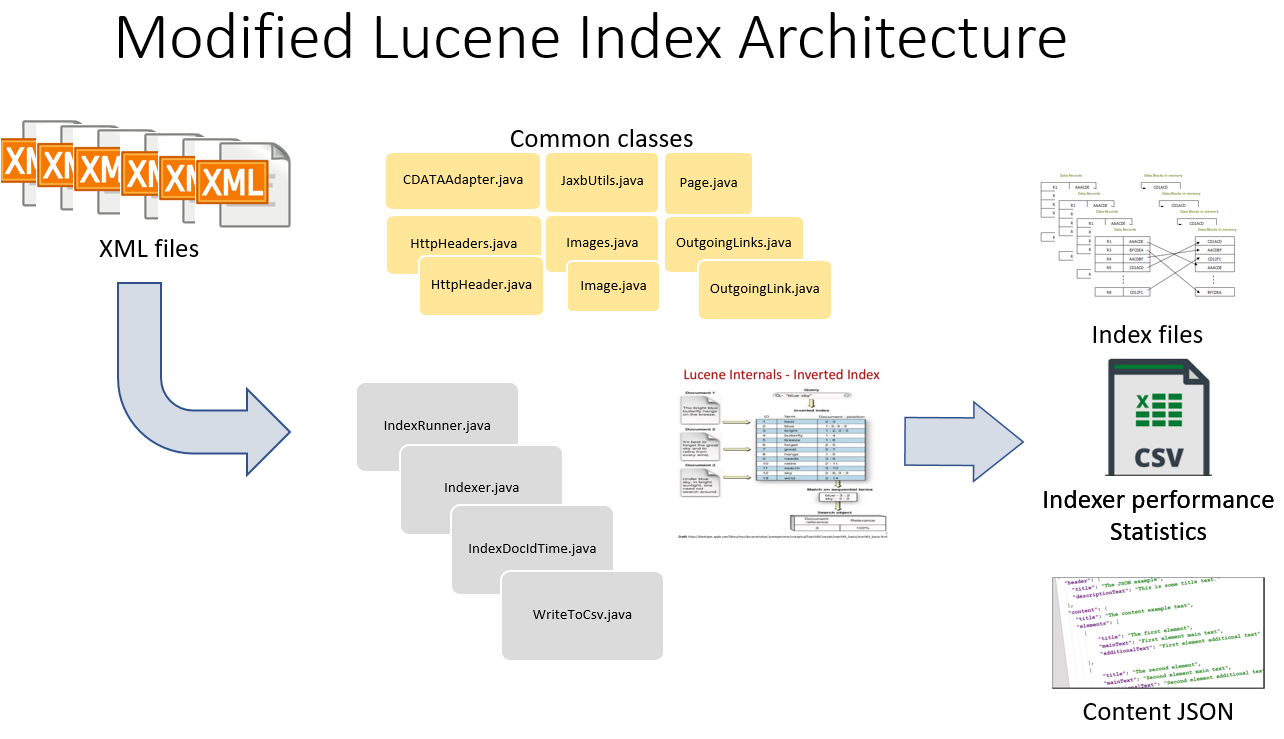


# Modified Lucene Indexing Architecture

## Modified Indexing Architecture

Indexing Architecture designed and developed in the previous phase of the project was retained for the most part. The only addition was to create Content JSON output from the Indexing process. Since the XML files from the crawling process had to be processed by the indexing logic, it was decided to take advantage of that and generate the Content JSON output that contains the actual content from the XML that can be used to generate the snippets for web search. Other option would have been to code new process just to extract the content portion of the document for snippet logic.

**Figure 3: Modified Lucene Index Architecture**

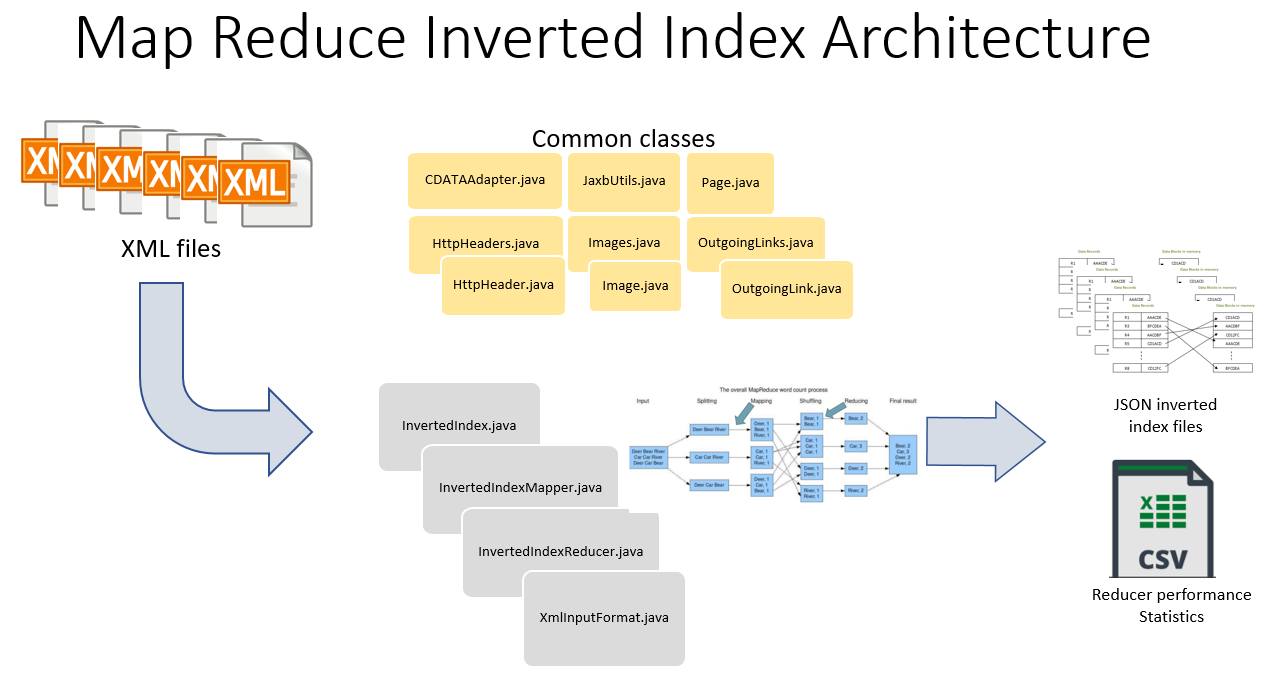


# Map Reduce Inverted Index Architecture and process

## Map Reduce Inverted Index Architecture

Map Reduce inverted index process was built in Java to read the XML files created by the Crawler and produce JSON output files with inverted index output. In addition, the process creates reducer processing time statistics by every keyword that is processed.

**Figure 4: Map Reduce Inverted Index Architecture**



## How was Hadoop used

Hadoop was used in the following manner to generate the Inverted Index using Map Reduce process:

1. XML files generated by Crawler were transferred and stored in HDFS in an input folder on HDFS
2. Map Reduce logic was executed to take the XML input file folder as input as one of the parameters and a folder to store the following files as output:
   1. JSON inverted index files
   2. Text file output with Reducer processing time by word which can be easily imported into a csv

**Mapper**:

Mapper code was developed to emit every keyword from “title” and “content” tags of each XML input file. So, tile and content tags from the XML were concatenated and provided as input to the mapper process. In addition, “docNo” from the XML input file was extracted and concatenated with the “position” of each keyword. So, the mapper was set up so that it can emit every “keyword” from “title” and “content” tags of each XML input file and the concatenated string of “docNo” and each “position” of the keyword being emitted.

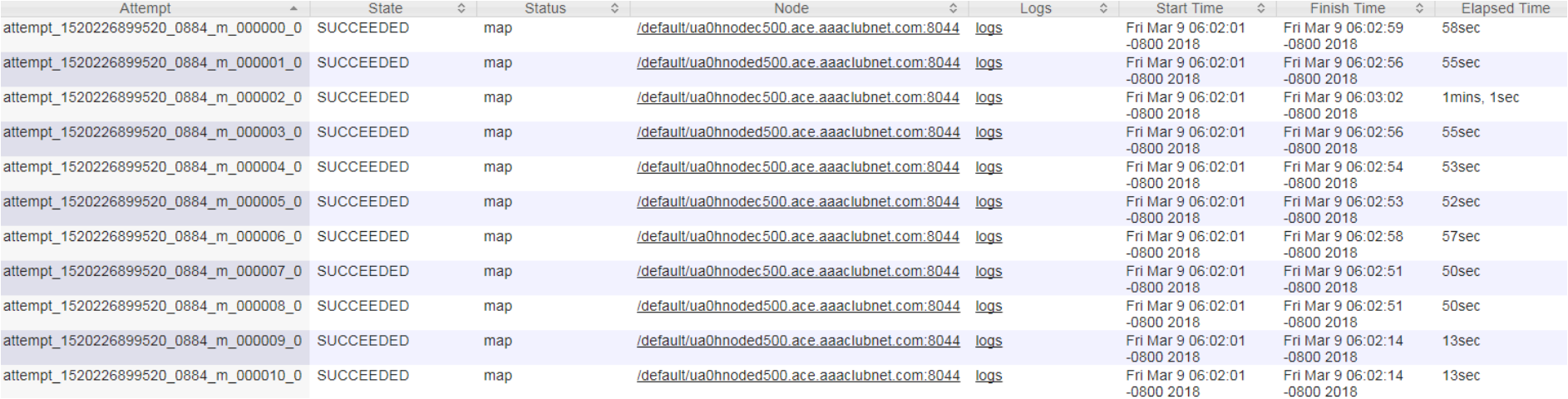
**Reducer**:

Reducer code was developed so that it does the combiner and reducer functionality by taking in the “keyword” and an interpretable text string of docNo and keyword position concatenated values. Logic within the reducer splits the docNo and Keyword concatenations, uses the keyword, docNo, collects all positions of occurrences of the keyword within each document and calculates the frequency of the keyword within each document. This enables the reducer to produce a JSON output that comprises of the following:

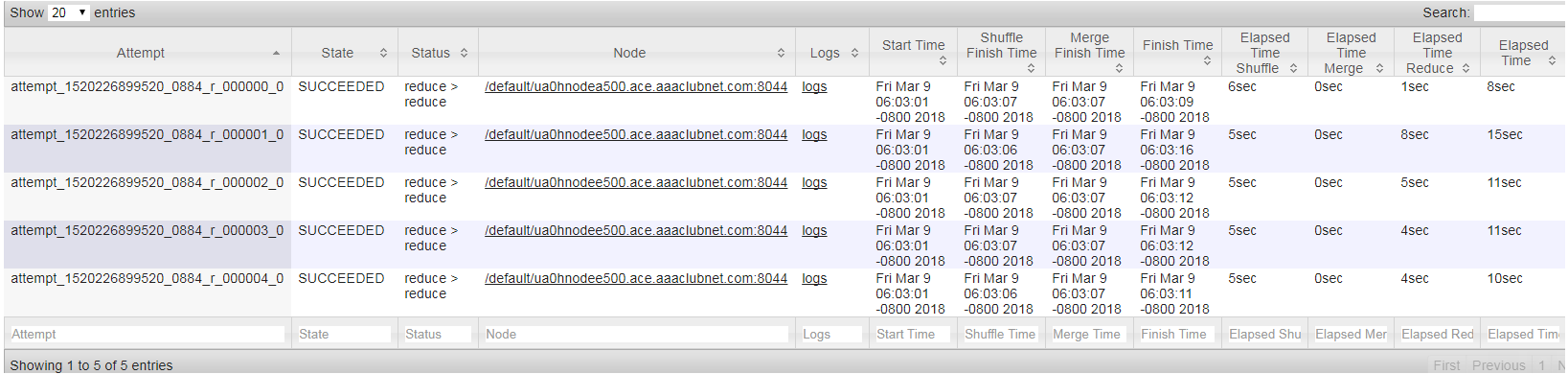
1. KEYWORD
2. DOCID
3. FREQUENCY
4. KEYWORDPOSITIONS

## Explanation of Indexes built by Hadoop

It was quickly found out during our testing on UCR Hadoop cluster that the resources available are limited. So, a decision was made to use an external cluster and explore options like using spark as the processing engine for the Map Reduce job instead of Map Reduce processing engine. 11 mappers were used by the Map reduce job to process the 5 GB worth of XML input data to create the inverted index and here are the processing times of the mapper tasks of the map reduce job that were gathered from the yarn logs:



5 Reducers were used by the Map reduce job to generate the JSON output files – so as a result 5 JSON output files got created – 1 per reducer. Here are the timings of the 5 reducer tasks of the map reduce job that were gathered from the yarn logs:



It was decided to not do stemming of the keywords since we wanted to show the results as is, based on the searches done on the search engine and be able to compare the output from Lucene indexing logic that was built in the previous phase where stopping was implemented but not stemming. Also, a decision was made to use the title and content tags of the XML that contained the richest data that is useful for querying. Based on a sampling of pages that were analyzed, it was found that a lot of pages in this domain had quite a few blogs and lots of links to advertisements and other pages which may not be useful for search engine. Our approach was to keep the search results as close to being as clean as possible for this project so that we can demonstrate the top few results.

Here is a sample of the JSON output of the Inverted Index from the Reducer:

{"KEYWORD":"Gimli,","DOCS":[{"DOCID":"1874145082","FREQUENCY":2,"KEYWORDPOSITIONS":[["1283","1329"]]},{"DOCID":"2065481290","FREQUENCY":1,"KEYWORDPOSITIONS":[["1341"]]},{"DOCID":"1732645740","FREQUENCY":1,"KEYWORDPOSITIONS":[["446"]]},{"DOCID":"1852423325","FREQUENCY":1,"KEYWORDPOSITIONS":[["81"]]},{"DOCID":"2131899548","FREQUENCY":1,"KEYWORDPOSITIONS":[["3039"]]},{"DOCID":"360598632","FREQUENCY":1,"KEYWORDPOSITIONS":[["172"]]},{"DOCID":"1310913358","FREQUENCY":1,"KEYWORDPOSITIONS":[["53"]]},{"DOCID":"1484412101","FREQUENCY":8,"KEYWORDPOSITIONS":[["20158","11427","1145","4094","18670","12508","11182","5897"]]},{"DOCID":"1712065679","FREQUENCY":1,"KEYWORDPOSITIONS":[["446"]]},{"DOCID":"893939465","FREQUENCY":1,"KEYWORDPOSITIONS":[["236"]]},{"DOCID":"2135761873","FREQUENCY":4,"KEYWORDPOSITIONS":[["993","258","63","862"]]},{"DOCID":"324695905","FREQUENCY":1,"KEYWORDPOSITIONS":[["364"]]},{"DOCID":"218193417","FREQUENCY":8,"KEYWORDPOSITIONS":[["11427","5897","1145","20158","11182","12508","4094","18670"]]},{"DOCID":"81289139","FREQUENCY":1,"KEYWORDPOSITIONS":[["276"]]},{"DOCID":"220495563","FREQUENCY":1,"KEYWORDPOSITIONS":[["68"]]},{"DOCID":"1732364870","FREQUENCY":1,"KEYWORDPOSITIONS":[["138"]]},{"DOCID":"1732316797","FREQUENCY":1,"KEYWORDPOSITIONS":[["1430"]]},{"DOCID":"1732036149","FREQUENCY":5,"KEYWORDPOSITIONS":[["727","852","788","919","1154"]]},{"DOCID":"1187723831","FREQUENCY":1,"KEYWORDPOSITIONS":[["37"]]},{"DOCID":"2108954692","FREQUENCY":1,"KEYWORDPOSITIONS":[["78"]]},{"DOCID":"1926759596","FREQUENCY":3,"KEYWORDPOSITIONS":[["1406","5126","2763"]]},{"DOCID":"388748971","FREQUENCY":1,"KEYWORDPOSITIONS":[["1328"]]},{"DOCID":"947793026","FREQUENCY":3,"KEYWORDPOSITIONS":[["1406","2763","5126"]]},{"DOCID":"1251249129","FREQUENCY":1,"KEYWORDPOSITIONS":[["1430"]]},{"DOCID":"1887385136","FREQUENCY":1,"KEYWORDPOSITIONS":[["2944"]]},{"DOCID":"721397402","FREQUENCY":1,"KEYWORDPOSITIONS":[["78"]]},{"DOCID":"667422545","FREQUENCY":1,"KEYWORDPOSITIONS":[["91"]]},{"DOCID":"405871197","FREQUENCY":1,"KEYWORDPOSITIONS":[["369"]]},{"DOCID":"861676550","FREQUENCY":2,"KEYWORDPOSITIONS":[["2457","136"]]},{"DOCID":"1833874339","FREQUENCY":1,"KEYWORDPOSITIONS":[["94"]]},{"DOCID":"549114421","FREQUENCY":1,"KEYWORDPOSITIONS":[["1395"]]},{"DOCID":"734171238","FREQUENCY":1,"KEYWORDPOSITIONS":[["3039"]]},{"DOCID":"1696231440","FREQUENCY":3,"KEYWORDPOSITIONS":[["5126","2763","1406"]]},{"DOCID":"1476606751","FREQUENCY":1,"KEYWORDPOSITIONS":[["238"]]},{"DOCID":"895834893","FREQUENCY":2,"KEYWORDPOSITIONS":[["672","1056"]]},{"DOCID":"471527739","FREQUENCY":1,"KEYWORDPOSITIONS":[["543"]]},{"DOCID":"1712096431","FREQUENCY":1,"KEYWORDPOSITIONS":[["446"]]},{"DOCID":"2001422573","FREQUENCY":3,"KEYWORDPOSITIONS":[["2257","1928","519"]]},{"DOCID":"723797502","FREQUENCY":1,"KEYWORDPOSITIONS":[["740"]]},{"DOCID":"1247778605","FREQUENCY":1,"KEYWORDPOSITIONS":[["216"]]},{"DOCID":"1234097847","FREQUENCY":2,"KEYWORDPOSITIONS":[["1384","1058"]]},{"DOCID":"1384679235","FREQUENCY":3,"KEYWORDPOSITIONS":[["2998","2895","6433"]]},{"DOCID":"811746577","FREQUENCY":1,"KEYWORDPOSITIONS":[["790"]]},{"DOCID":"1018030620","FREQUENCY":1,"KEYWORDPOSITIONS":[["927"]]},{"DOCID":"256406693","FREQUENCY":2,"KEYWORDPOSITIONS":[["1957","1609"]]},{"DOCID":"2135122052","FREQUENCY":2,"KEYWORDPOSITIONS":[["1957","1609"]]},{"DOCID":"719280390","FREQUENCY":1,"KEYWORDPOSITIONS":[["161"]]},{"DOCID":"2141474916","FREQUENCY":3,"KEYWORDPOSITIONS":[["1406","5126","2763"]]},{"DOCID":"1728159712","FREQUENCY":2,"KEYWORDPOSITIONS":[["1957","1609"]]},{"DOCID":"162833916","FREQUENCY":1,"KEYWORDPOSITIONS":[["3814"]]},{"DOCID":"425182840","FREQUENCY":3,"KEYWORDPOSITIONS":[["1279","3557","840"]]},{"DOCID":"1321797369","FREQUENCY":1,"KEYWORDPOSITIONS":[["446"]]},{"DOCID":"178553135","FREQUENCY":3,"KEYWORDPOSITIONS":[["2763","1406","5126"]]},{"DOCID":"1107846769","FREQUENCY":1,"KEYWORDPOSITIONS":[["4827"]]},{"DOCID":"1526352937","FREQUENCY":1,"KEYWORDPOSITIONS":[["446"]]},{"DOCID":"601400221","FREQUENCY":1,"KEYWORDPOSITIONS":[["1395"]]},{"DOCID":"340436060","FREQUENCY":2,"KEYWORDPOSITIONS":[["2005","2060"]]},{"DOCID":"1663096612","FREQUENCY":1,"KEYWORDPOSITIONS":[["72"]]},{"DOCID":"1892446768","FREQUENCY":1,"KEYWORDPOSITIONS":[["892"]]},{"DOCID":"2135670666","FREQUENCY":2,"KEYWORDPOSITIONS":[["90","942"]]},{"DOCID":"439632838","FREQUENCY":1,"KEYWORDPOSITIONS":[["80"]]},{"DOCID":"338745657","FREQUENCY":3,"KEYWORDPOSITIONS":[["519","2257","1928"]]},{"DOCID":"72388058","FREQUENCY":3,"KEYWORDPOSITIONS":[["2763","1406","5126"]]},{"DOCID":"298855282","FREQUENCY":3,"KEYWORDPOSITIONS":[["2895","6433","2998"]]},{"DOCID":"644220208","FREQUENCY":2,"KEYWORDPOSITIONS":[["656","501"]]},{"DOCID":"1548306982","FREQUENCY":3,"KEYWORDPOSITIONS":[["6433","2998","2895"]]},{"DOCID":"400231229","FREQUENCY":1,"KEYWORDPOSITIONS":[["459"]]},{"DOCID":"2137430377","FREQUENCY":2,"KEYWORDPOSITIONS":[["298","231"]]},{"DOCID":"471654591","FREQUENCY":1,"KEYWORDPOSITIONS":[["543"]]},{"DOCID":"1533234861","FREQUENCY":6,"KEYWORDPOSITIONS":[["1962","4075","565","478","2750","460"]]},{"DOCID":"1624476607","FREQUENCY":1,"KEYWORDPOSITIONS":[["290"]]},{"DOCID":"2005607565","FREQUENCY":1,"KEYWORDPOSITIONS":[["927"]]},{"DOCID":"2041162976","FREQUENCY":1,"KEYWORDPOSITIONS":[["3039"]]},{"DOCID":"1569741882","FREQUENCY":3,"KEYWORDPOSITIONS":[["2895","6433","2998"]]},{"DOCID":"602384822","FREQUENCY":1,"KEYWORDPOSITIONS":[["927"]]},{"DOCID":"2122499805","FREQUENCY":1,"KEYWORDPOSITIONS":[["927"]]},{"DOCID":"1036755451","FREQUENCY":1,"KEYWORDPOSITIONS":[["446"]]},{"DOCID":"1938769540","FREQUENCY":3,"KEYWORDPOSITIONS":[["1279","3557","840"]]},{"DOCID":"355666508","FREQUENCY":1,"KEYWORDPOSITIONS":[["1341"]]},{"DOCID":"1657323206","FREQUENCY":2,"KEYWORDPOSITIONS":[["1056","672"]]},{"DOCID":"2032179815","FREQUENCY":8,"KEYWORDPOSITIONS":[["12508","4094","1145","5897","20158","11427","18670","11182"]]},{"DOCID":"1054175143","FREQUENCY":1,"KEYWORDPOSITIONS":[["89"]]},{"DOCID":"1845919315","FREQUENCY":1,"KEYWORDPOSITIONS":[["152"]]},{"DOCID":"580657480","FREQUENCY":1,"KEYWORDPOSITIONS":[["236"]]},{"DOCID":"856076010","FREQUENCY":1,"KEYWORDPOSITIONS":[["3039"]]},{"DOCID":"125957991","FREQUENCY":1,"KEYWORDPOSITIONS":[["892"]]},{"DOCID":"2060805339","FREQUENCY":1,"KEYWORDPOSITIONS":[["740"]]},{"DOCID":"903538516","FREQUENCY":2,"KEYWORDPOSITIONS":[["1957","1609"]]},{"DOCID":"757669432","FREQUENCY":1,"KEYWORDPOSITIONS":[["78"]]},{"DOCID":"1961789071","FREQUENCY":1,"KEYWORDPOSITIONS":[["369"]]},{"DOCID":"724823120","FREQUENCY":1,"KEYWORDPOSITIONS":[["122"]]},{"DOCID":"1491369779","FREQUENCY":1,"KEYWORDPOSITIONS":[["790"]]},{"DOCID":"2015525605","FREQUENCY":2,"KEYWORDPOSITIONS":[["231","298"]]},{"DOCID":"1331080676","FREQUENCY":3,"KEYWORDPOSITIONS":[["289","180","201"]]},{"DOCID":"1709782925","FREQUENCY":1,"KEYWORDPOSITIONS":[["1676"]]},{"DOCID":"1732338900","FREQUENCY":1,"KEYWORDPOSITIONS":[["1430"]]},{"DOCID":"1578364301","FREQUENCY":1,"KEYWORDPOSITIONS":[["790"]]},{"DOCID":"368399360","FREQUENCY":1,"KEYWORDPOSITIONS":[["73"]]},{"DOCID":"1121349720","FREQUENCY":5,"KEYWORDPOSITIONS":[["788","727","852","919","1154"]]},{"DOCID":"1732189945","FREQUENCY":1,"KEYWORDPOSITIONS":[["1430"]]},{"DOCID":"1732332922","FREQUENCY":3,"KEYWORDPOSITIONS":[["180","201","289"]]},{"DOCID":"1732189949","FREQUENCY":3,"KEYWORDPOSITIONS":[["2763","5126","1406"]]},{"DOCID":"895130389","FREQUENCY":1,"KEYWORDPOSITIONS":[["504"]]},{"DOCID":"1359433236","FREQUENCY":1,"KEYWORDPOSITIONS":[["1206"]]},{"DOCID":"549796292","FREQUENCY":2,"KEYWORDPOSITIONS":[["2457","136"]]},{"DOCID":"89525002","FREQUENCY":3,"KEYWORDPOSITIONS":[["1279","840","3557"]]},{"DOCID":"2145355071","FREQUENCY":8,"KEYWORDPOSITIONS":[["18670","11427","20158","11182","4094","1145","5897","12508"]]},{"DOCID":"1388202848","FREQUENCY":2,"KEYWORDPOSITIONS":[["1957","1609"]]},{"DOCID":"961531376","FREQUENCY":1,"KEYWORDPOSITIONS":[["2139"]]},{"DOCID":"872739755","FREQUENCY":1,"KEYWORDPOSITIONS":[["790"]]},{"DOCID":"102115292","FREQUENCY":1,"KEYWORDPOSITIONS":[["218"]]},{"DOCID":"1306026993","FREQUENCY":1,"KEYWORDPOSITIONS":[["4827"]]},{"DOCID":"2134804829","FREQUENCY":1,"KEYWORDPOSITIONS":[["324"]]},{"DOCID":"888531196","FREQUENCY":1,"KEYWORDPOSITIONS":[["1206"]]}]}

## Explanation of how Hadoop-created inverted index was used to calculate ranking

<Jorge / Hovanes to fill this section - it is related to ranking>

JSON output file (sample of which is shown above) was loaded into inverted index collection in a MongoDB database that we set up for the project. Python code was executed on the inverted index collection to do BM25 ranking. The code takes in each term from the query and looks it up on the inverted index stored on MongoDB and retrieves each document that each query term is present it. Then, it calculates BM25 for each of these documents, running a loop over each query term again for the addends of the BM25 calculation. While accessing the inverted index collection in MongoDB, this code also retrieves the position of the query term in each document to pass to the snippet generation code which will be discussed in the next section. This information is stored in a matrix of results to be sent to the UI which will present the results after being sorted by BM25 score. An illustration of some of the BM25 code is below:

N = self.N # documents in corpus  
k1 = 1.2  
k2 = 100  
b = 0.75  
ri = 0  
R = 0  
avdl = 450 # average document length

for d in kw\_doc['DOCS'][0:19]: # run bm25 for top 20 doc frequencies

bm25 = 0 # reset bm25 value for each new document  
 doc\_id = d['DOCID'] # string  
 pos = d['KEYWORDPOSITIONS'][0][0] # position of word in the document for the snippet  
 content = self.get\_content\_for\_doc(doc\_id)  
 c = nltk.word\_tokenize(content)  
 dl = len(c) # document length

for t in filtered: # calculate addend of bm25 for each query term   
 x = [m.start() for m in re.finditer(t, content)]  
 fi = len(x)  
 K = k1 \* ((1 - b) + b \* dl / avdl)  
 bm25 = math.log1p(((ri + 0.5) / (R - ri + 0.5)) / ((ni - ri + 0.5) / (N - ni - R + ri + 0.5)))

\* (k1 + 1) \* fi / (K + fi) \* (k2 + 1) \* qfi / (k2 + qfi)  
 bm25 += bm25 # sum the tbm25 for each term to get bm25 for the query  
 snip = self.get\_snip\_for\_doc(c, pos) # fetch the snippet  
 results.score = bm25   
 results.docId = d  
 results.snippet = snip  
 results.docNo = content['docno']  
 results.title = content['doctitle']  
 results.url = content['url']  
 results.append(results.data())

## Explanation of how snippets are generated

To retrieve snippets for display in the search results during the MapReduce query option, we converted the crawled XML to JSON and uploaded those files to MongoDB. Our Python code for the search engine is easily able to retrieve this information from MongoDB. For each document in the query results, we retrieve the main content of that page and select from it the range of text that contains the query term for context. The calculation of this range is illustrated in the below Python code, where the lower and upper bounds of the term positions are calculated. We chose to return a snippet length of 20, and to have the query term be in the middle of that snippet if possible for context, but the calculation takes into account cases where the query term is in the beginning of the document.

content = self.get\_content\_for\_doc(doc\_id) # retrieve content from MongoDB “snippets” collection

pos = self.ii\_coll.find({'KEYWORD': t})['DOCS']['KEYWORDPOSITIONS'][0][0] # term position from MongoDB “inverted index” collection

sl = 20 # snippet length  
if p >= sl:  
 lb = p - sl # lower bound position  
 rsl = 0  
else:  
 lb = p  
 rsl = sl - p  
  
ub = p + sl + rsl # upper bound position  
s = content[lb:ub] # snippet to be sent to results matrix

## Explanation of how Lucene index was used to return results

<Jorge / Hovanes to fill this section - it is related to search engine>

Explain how you use the Lucene index to return results. E.g., what options did you use with the QueryParser? How were multiple fields in the Lucene index used with the QueryParser? 🡪 project.docx statement

# Limitations of the system

<Jorge / Hovanes to fill this section >

# Obstacles and solutions

* Wanted to use HBase to store the inverted index output for easy querying by the web application. After checking with TA it was found that HBase is not available on the UCR cluster and that the UCR cluster is minimalistic in the features that are available. So, a decision was made to push the XML data into another Hadoop cluster and run the map reduce process there so that we can take advantage of spark processing engine.
* Also, the team made the decision to use MongoDB as the database to store the snippet JSON data and Inverted Index JSON data.
* Architecture of the system was adjusted to use MongoDB and python to set up the search engine and to query the collections in MongoDB for processing.

# Index Process time comparison between Lucene and Map Reduce Inverted Index

**Figure 5 - Lucene vs MapReduce Performance**

The figure above illustrates the performance of Lucene in green and the performance of MapReduce in blue at different stages of the indexing process. MapReduce started off faster than Lucene and its speed advantage only grew more pronounced as the collection size grew.

**Figure 2: Lucene vs MapReduce vs Collection Size**

The above figure illustrates the performance of each indexing system as a function of collection size. Again, it is clear given these statistics that Hadoop is a faster option for indexing a dataset of this size.

# How to deploy the system

<other steps to deploy the system to be filled in by Jorge and Hovanes>

**Steps to execute MapReduce code:**

* Deploy the attached code from src zip file:



* Compile the code to create the jar files or use the jar files provided and copy them to the folder from where the code will be executed
* Copy Libjars java-json.jar,json-simple-1.1.jar to the same location as that of MapReduce jar
* Input folder = /tmp/XML, output folder=/tmp/XML\_out
* Hadoop jar inverted\_index\_with\_time.jar edu.ucr.cs242.InvertedIndexMain -libjars java-json.jar,json-simple-1.1.jar /tmp/XML /tmp/XML\_out

<other steps to deploy the system to be filled in by Jorge and Hovanes>

# Screenshots of the system in action

<Jorge / Hovanes to fill this section >