CSC 8712

A framework for building inverted index on textual data

- Gaurav Jangir, Rajath CS, Moulika Achhe

Introduction:

Background:

- Data is growing exponentially. There is a need for enhancements in techniques for processing and querying of textual data.
- Searching and querying of records are the two bottleneck.

Evolution of searching and querying from large corpus of data:

1. RAID (Redundant Array of Independent Disks).
2. Hashing(Equality and Membership Queries).
3. Indexing(secondary access paths).

Introduction:

What is an Inverted Index:

- An inverted index is the mapping of words to documents. A storage of a list of words/literals and the documents in which they appear.
- An inverted index can also store the occurrence(frequencies) of words or weights.

Inverted Index vs Database Index:

- A list of documents(block of records) and the words that it contains.
- Table of Contents(TOC) vs Sections(End of a book).

Inverted Index vs Database Index

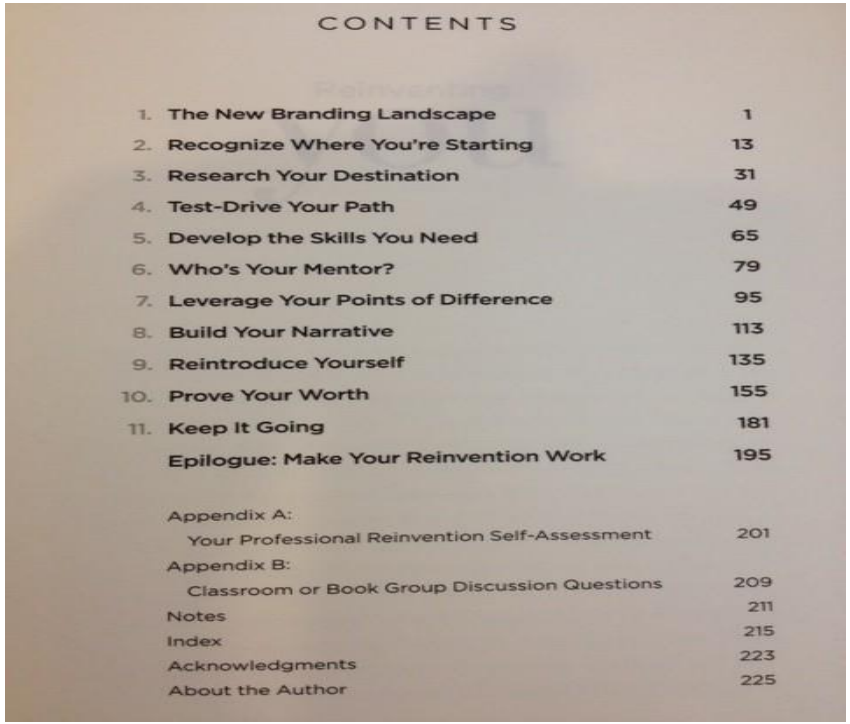
—Index—

–A–

about the author 128, 132, 412
account info 295
active table of contents 34, 120-124, 238-239, 285-286, 354, 366, 370
ACX 465-467
Adobe 506
advertising 434, 439-449
age 312
aggregator 17-18, 322
alignment 68, 101-103, 105-106, 229-230, 261-262, 353-354, 380, 389
Alt codes 39
Amazon Associates 415
Amazon Follow 430, 437, 480
Amazon Giveaway 436-439
Amazon Marketing Services (AMS) 439-449
Android 167-169, 171, 371-375
apostrophe 40, 42-44
app 141-142
Apple 169, 342, 372, 506
automatic renewal 327-329, 341, 343
Automatically Update 73-75, 94, 144
AZK 371

–B–

back matter 124-129
background 47, 93, 181, 184, 192-193, 246, 252-253, 355, 370, 385, 390
bank information 295
Barnes & Noble 506
biography 128, 132, 410
black 47, 93, 184, 192, 252-253, 355, 370, 385, 390
Blackberry 372-373
blank line 27-28, 110, 112-114, 276-277, 284-285, 385
blank page 354, 385-386
block indent 50, 52, 67, 82, 106-107, 234-235
blog 411, 429, 479
Blogger 429
bloggers 327, 430
blurb 300-306, 364, 406, 411-412, 417, 477
blurry 162-164, 172, 175, 193, 246, 387, 389
body text 66, 68, 79-82, 92-94, 115, 233-235



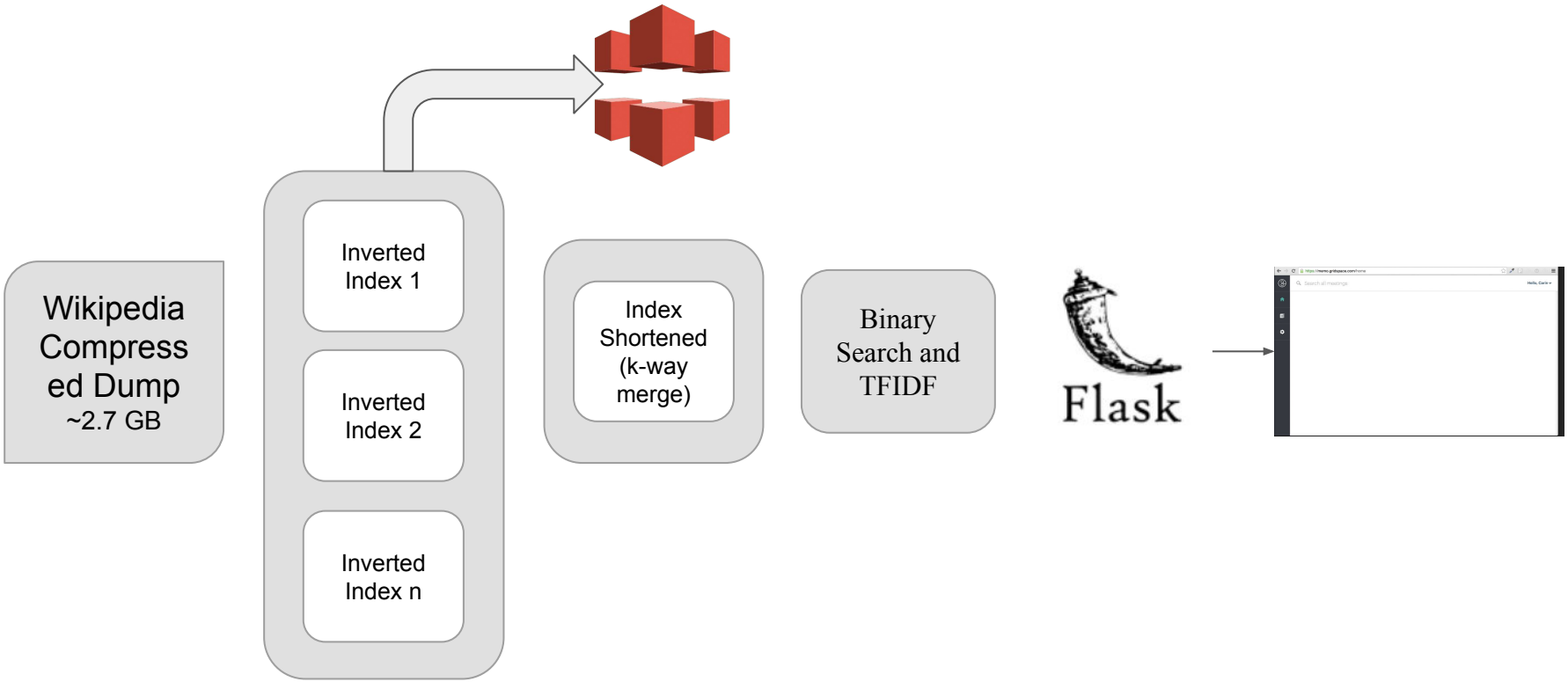
CONTENTS

1. The New Branding Landscape 1
2. Recognize Where You're Starting 13
3. Research Your Destination 31
4. Test-Drive Your Path 49
5. Develop the Skills You Need 65
6. Who's Your Mentor? 79
7. Leverage Your Points of Difference 95
8. Build Your Narrative 113
9. Reintroduce Yourself 135
10. Prove Your Worth 155
11. Keep It Going 181

Epilogue: Make Your Reinvention Work 195

Appendix A:
Your Professional Reinvention Self-Assessment 201
Appendix B:
Classroom or Book Group Discussion Questions 209
Notes 211
Index 215
Acknowledgments 223
About the Author 225

System Overview:



Methodology:

I. Indexing

- We pre-process the whole data by removing stopwords and applying stemming.
- We create an inverse map of document id to wiki page title.
- The key will be a word and the value will be a list of file_ids (wikipage_id) and how many times this word occurred in that file in title, body, infobox, references etc.
    - apple : {"file_id t2 b3 r4", "file_id1 t2 b3 r4", "file_id2 t2 b3 r4"}.
    - banana : {"file_id t4 b6 r3"}.
- As this inverted index can grow very large, we try to do it in a batch wise manner and create an index file for this chunk.

Methodology:

II. Merging Indices:

- We merge these sorted index files into a single file.
- We use K-way merge with min-heap to merge as we cannot load all the data into memory at once.
- Min-heap will help us in merging the sorted files in a reduced complexity of NlogK as the least value is always at the root node.

Main Idea:

Min-heap will return the smallest element from a collection of k-smallest elements from k-sorted lists.

Accordingly, we merge the sorted files in our case in a time complexity of O(NlogK).

Methodology:

III. Querying and Ranking:

- Given a sequence of words, we need to filter the documents, and ranking is done on these documents.
- Field queries such as “Title: Mahatma Body: India” are also supported.
- Based on the filtered documents, we score each document with a cumulative sum of TF-IDF for each word.
- For each word in the query, we search the index file by doing a binary search.

Main Idea: A binary search will help in finding the relevant querying the ranked(sorted) elements in $O(\log_2 N)$.

Tech Stack:

XML, Python, Flask, HTML, CSS, Amazon S3.

- For purpose of searching, in most cases we use engines like Solr/elasticsearch.
- Indexing also involves implementing some indexing technique. Here we will build index and search from scratch.

Dataset:

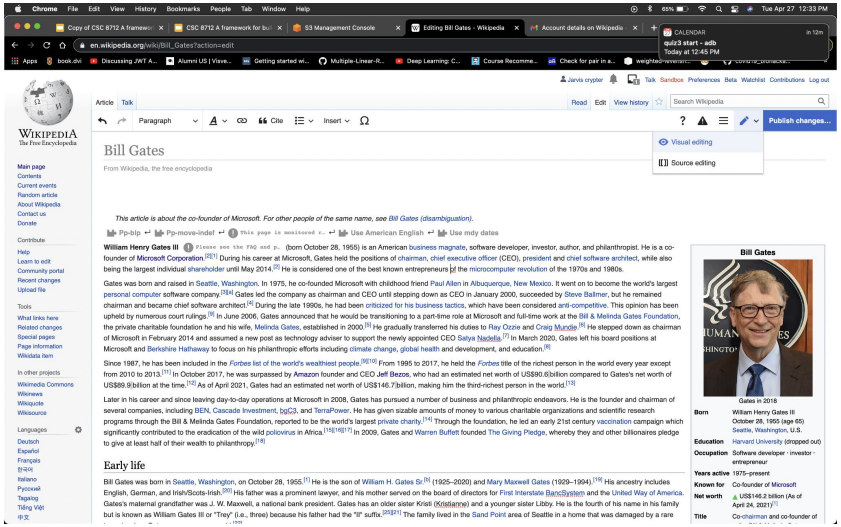
- Wikipedia XML Dump.
  - https://meta.wikimedia.org/wiki/Data_dump_torrents#English_Wikipedia
  - ~2.7GB(zipped)
- Ever growing dataset.
  - Wikipedia ~2500 articles/day, growing at terabytes rate.
- Various Formats: JSON,XML, RDF.
- Open-Source Dataset.
- Articles are separated by a pair of tags as in *<page>contents</page>*. The title of each article is marked by *"<title>Name of the article</title>"*
  - Links to other Wikipedia articles are of the form "[[Name of other article]]".

Implementation: Data Pre-processing

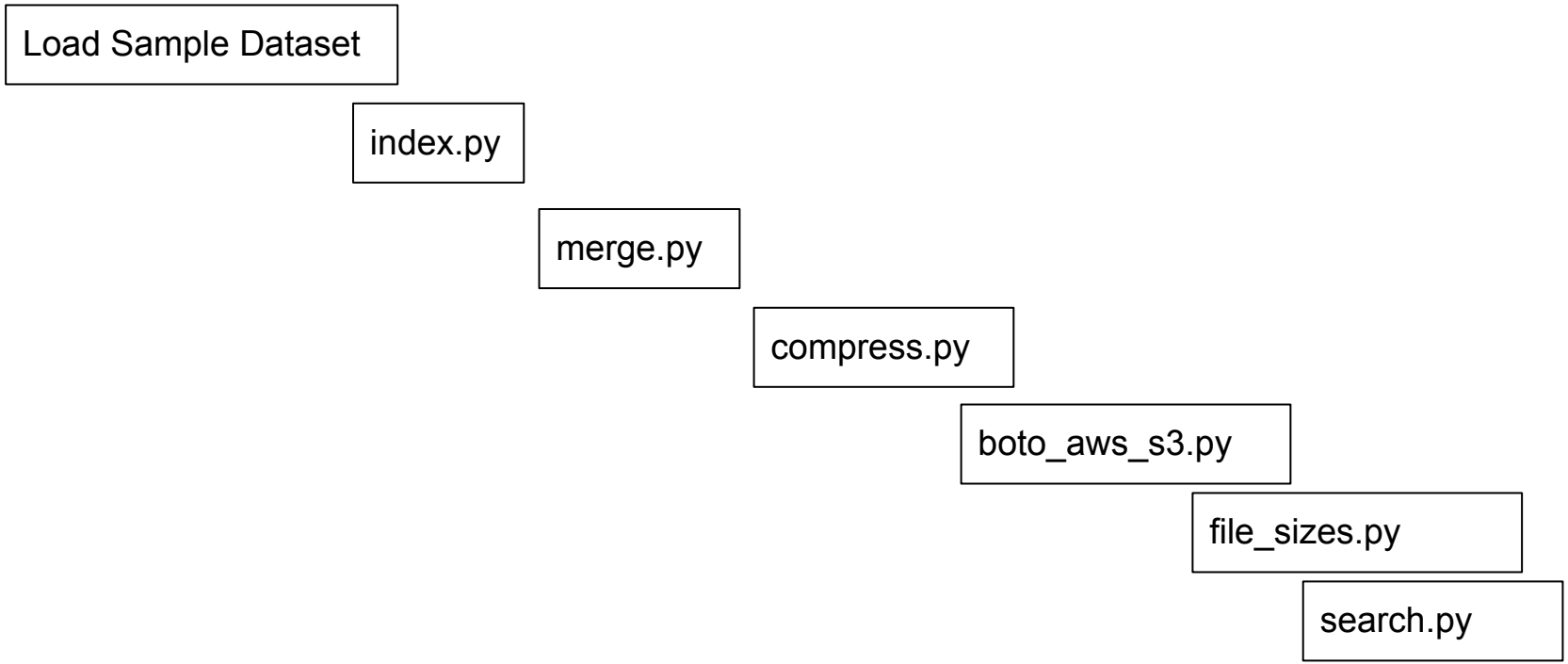
Bill Gates

From Wikipedia, the free encyclopedia

```
{{Short description|American business magnate and philanthropist}}
{{about|the co-founder of Microsoft|other people of the same name|Bill Gates (disambiguation)}}
{{pp-blp|small=yes}}
{{pp-move-indef}}
<!-- This page is monitored regularly for nonsense and vandalism. People who vandalize biographies about living people will be blocked from editing. If you would like to experiment with
Wikipedia, please copy the following address into your browser's address bar: https://en.wikipedia.org/wiki/ and it will take you to a page where new users can try out the editing
features! -->
{{Use American English|date=March 2020}}
{{Use mdy dates|date=April 2021}}
{{Infobox person
| name          = Bill Gates
| post-nominals = <!-- Please see the FAQ and previous discussions on the article's talk page. Including post-nominals at the top of the infobox would give them an undue prominence
in an article about an American who has no close association with the British realm. -->
| image         = Bill Gates 2018.jpg<!-- Consensus in talk page for this image. If you want to change it go to talk page -->
| alt           = Head and shoulders photo of Bill Gates
| caption       = Gates in 2018
| birth_name    = William Henry Gates III
| birth_date    = {{birth date and age|1955|10|28}}
| birth_place   = [[Seattle]], [[Washington (state)|Washington]], U.S.
| education     = [[Harvard University]] (dropped out)
| alma_mater    = <!--If there is anything that questions status of a subject as an alumnus of a university, it is not included in the alma mater/education section (i.e. reference
Natalie Portman vs. Mark Zuckerberg)—both attended Harvard, one graduated, the other did not.-->
| known_for     = Co-founder of [[Microsoft]]
| occupation    = {{hlist|Software developer|investor|entrepreneur}}
| net_worth     = {{up}} US$146.2 billion ({{As of|2021|04|24|df=US}})<ref name="Bloomberg">{{cite web|url=https://www.bloomberg.com/billionaires/profiles/william-h-gates/
|title=Bloomberg Billionaires Index: Bill Gates |publisher=Bloomberg |access-date=April 24, 2021}}</ref>
| years_active  = 1975–present
| title         = {{indented plainlist|
* Co-[[chairman]] and co-founder of the [[Bill & Melinda Gates Foundation]]
* Chairman and founder of [[Branded Entertainment Network]]
* Chairman and founder of [[Cascade Investment]]
* Chairman and co-founder of [[TerraPower]]
* [[Technical advisor|Technology advisor]] of Microsoft}}
| boards        =
| spouse        = {{marriage|[[Melinda Gates|Melinda French]]|January 1, 1994}}
| children      = 3
```



Execution Flow



index.py > ...

```python
import xml

import nltk
nltk.download('stopwords')

from nltk.corpus import stopwords
from datetime import datetime
import os
import pickle
import json
import re
from nltk import stem
from collections import Counter
from collections import defaultdict
import xml.sax as sax

LEN_INFOBOX  = len('{{infobox')
LEN_CATEGORY = len('[[category:')
LEN_EXTERNS  = len('==external links==')

TAG_RE = re.compile(r'<[^>]+>')
infoBoxRegex = re.compile(r'(\{\{infobox(.|\n)*?\}\}\n)(?:[^\|])')
categoryRegex = re.compile(r'\[\[category:.*\]\]\n')
externalsRegex = re.compile(r'==external links==\n(.|\n)*?\n\n')
reftagRegex =  re.compile(r'<ref(.|\n)*?</ref>')
refsRegex = re.compile(r'(==references==(.|\n)*?\n)(==|\{\{DEFAULTSORT)')
stop_words = set(stopwords.words('english'))
stemmer = stem.PorterStemmer()
MIN_STOP_LENGTH = 3
MAX_STOP_LENGTH = 15
alphabet_lis = 'abcdefghijklmnopqrstuvwxyz'

# punctuation='¡©!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~0123456789'
dert = [ele for ele in list(range(0,1_1141_111)) if ele not in list(range(97,123))]
```

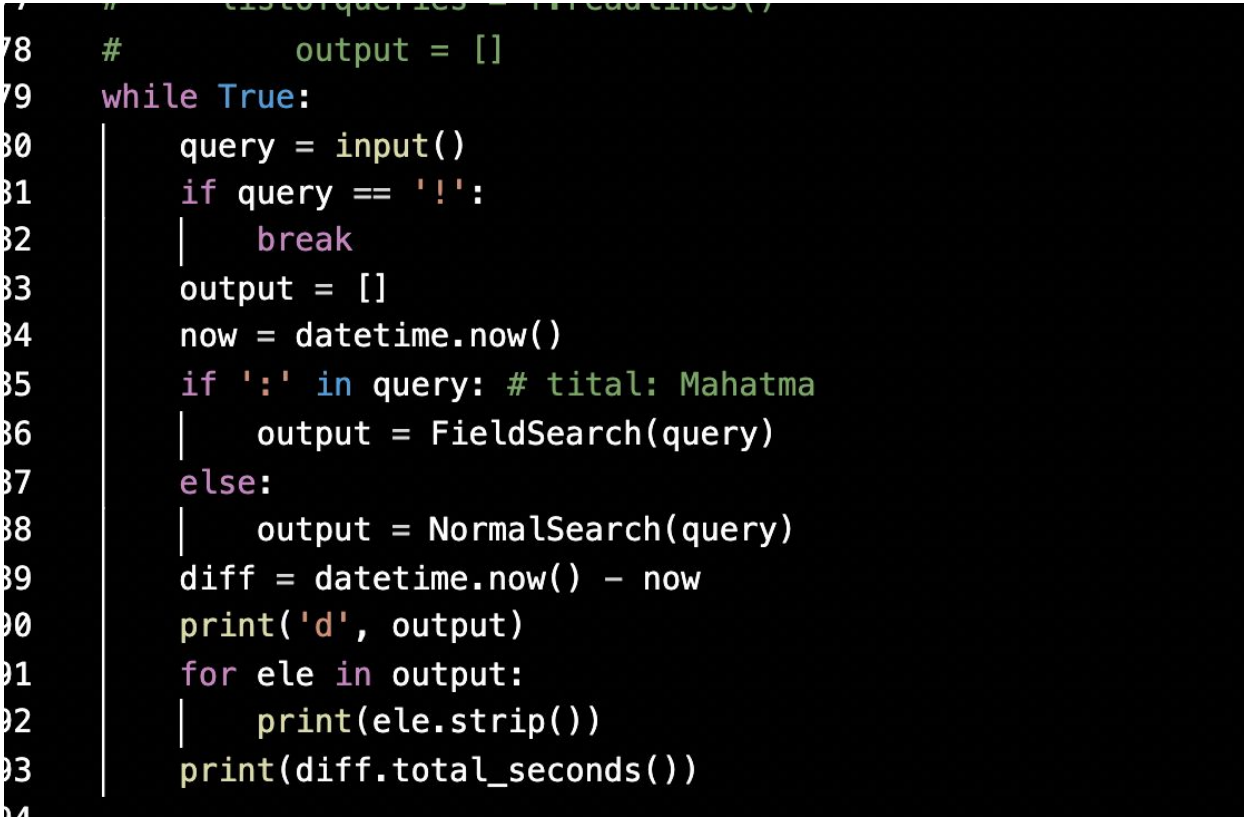
index.py

```
parser = xml.sax.make_parser()
handler = WikiHandler()
hellp = parser.setContentHandler(handler)
now = datetime.now()
paths = ['/Applications/Final/data/shortest.xml','/Applications/Final/data/Infosys.xml','/Applications/Final/data/Georgia_S
for path in paths:
    parser.parse(path)
diff = datetime.now() - now
print(diff.total_seconds())
```

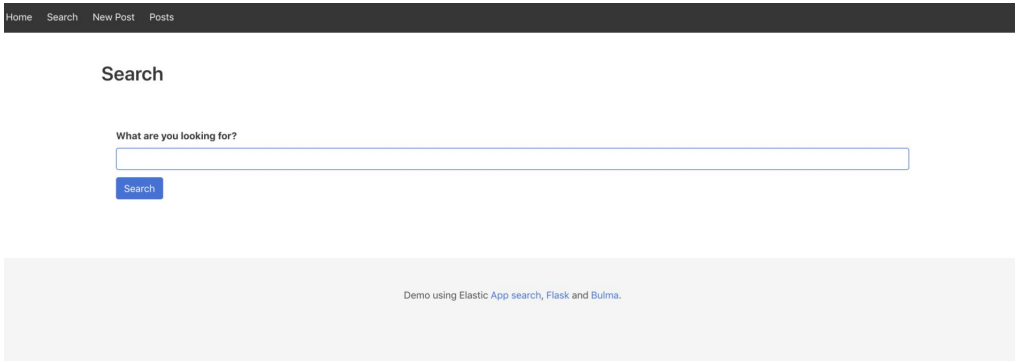
merge.py

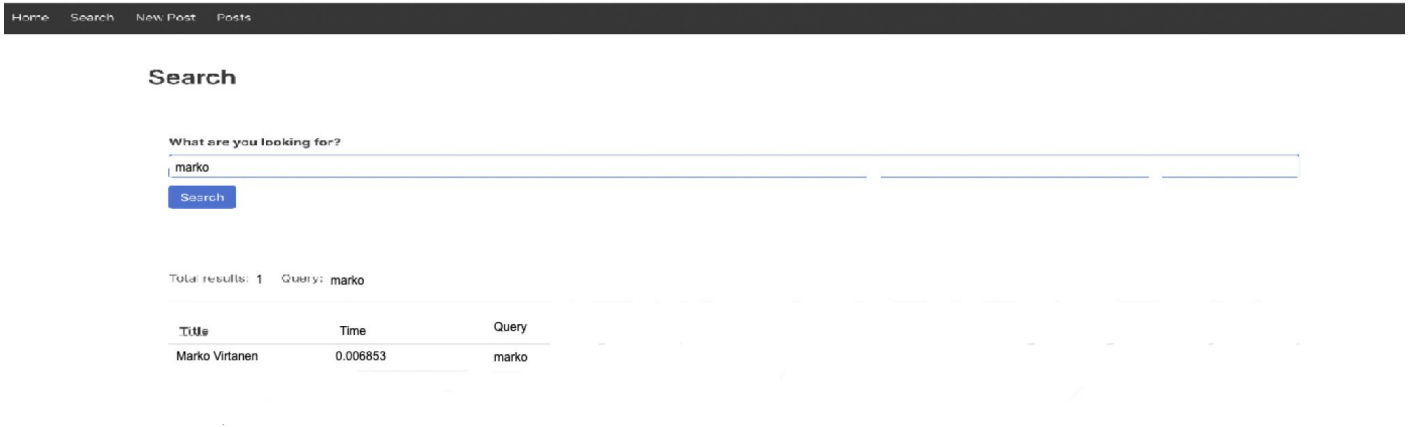
```
def merge(input_files):
    _heap = []
    data = ''
    _output_file = open('./index_m','w+')
    if True:
        open_files = []
        [open_files.append(open(os.path.join('./buggi_dir/',file__), 'r')) for file__ in input_files]
        # enqueue the pair (key,sent,f) using the first value as priority key
        for file__ in open_files:
            heapq.heappush(_heap,tuplify(file__))
        while(_heap):
            smallest = heapq.heappop(_heap)
            data += smallest[1]
            _output_file.write(smallest[1])
            # read next line from current file
            tuple__ = tuplify(smallest[2])
            # check that this file has not ended
            if(len(tuple__[1]) != 0):
                heapq.heappush(_heap, tuple__)
```

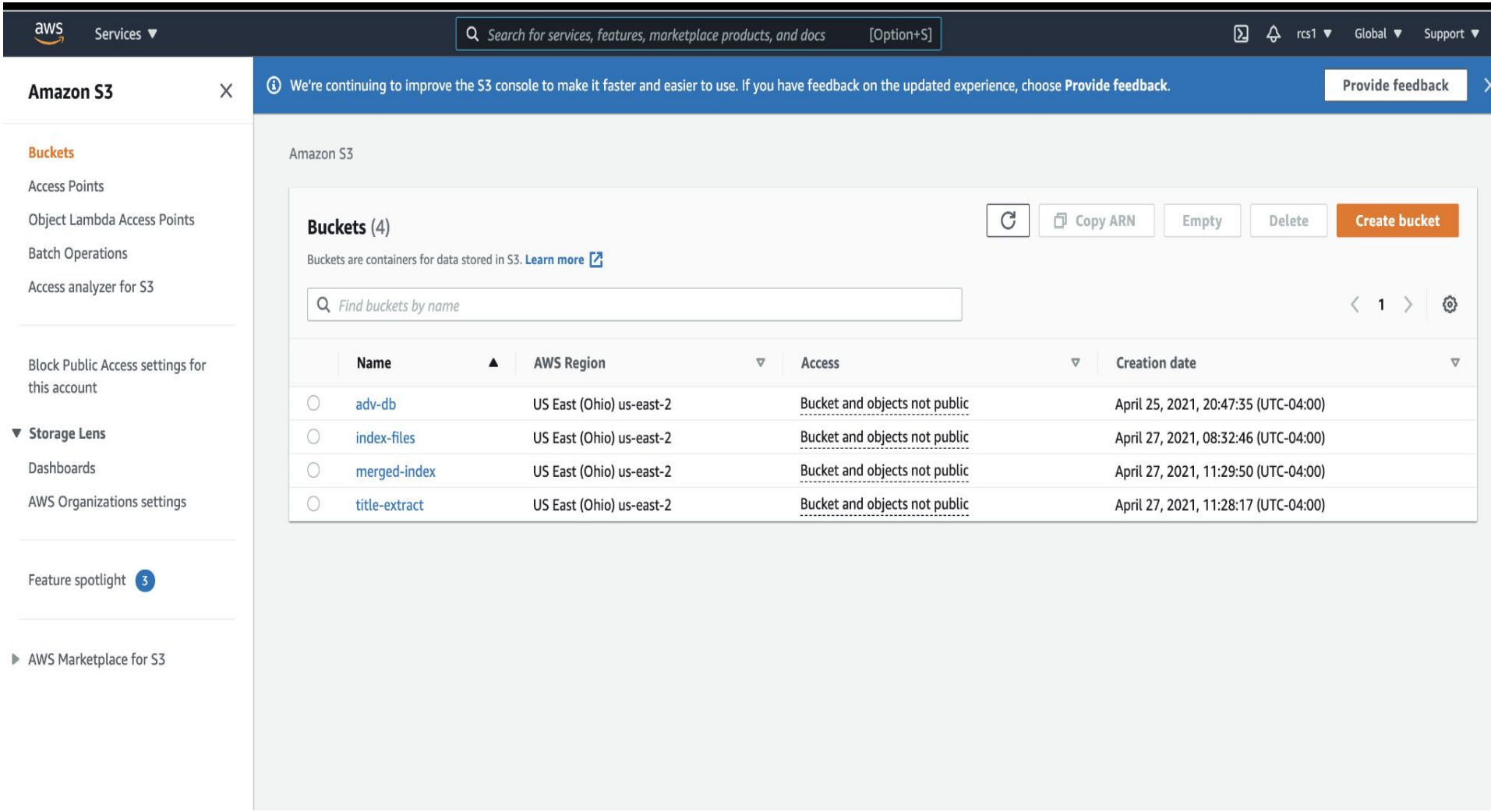
search.py


#   output = []
while True:
    query = input()
    if query == '!':
        break
    output = []
    now = datetime.now()
    if ':' in query: # tital: Mahatma
        output = FieldSearch(query)
    else:
        output = NormalSearch(query)
    diff = datetime.now() - now
    print('d', output)
    for ele in output:
        print(ele.strip())
    print(diff.total_seconds())

Application View





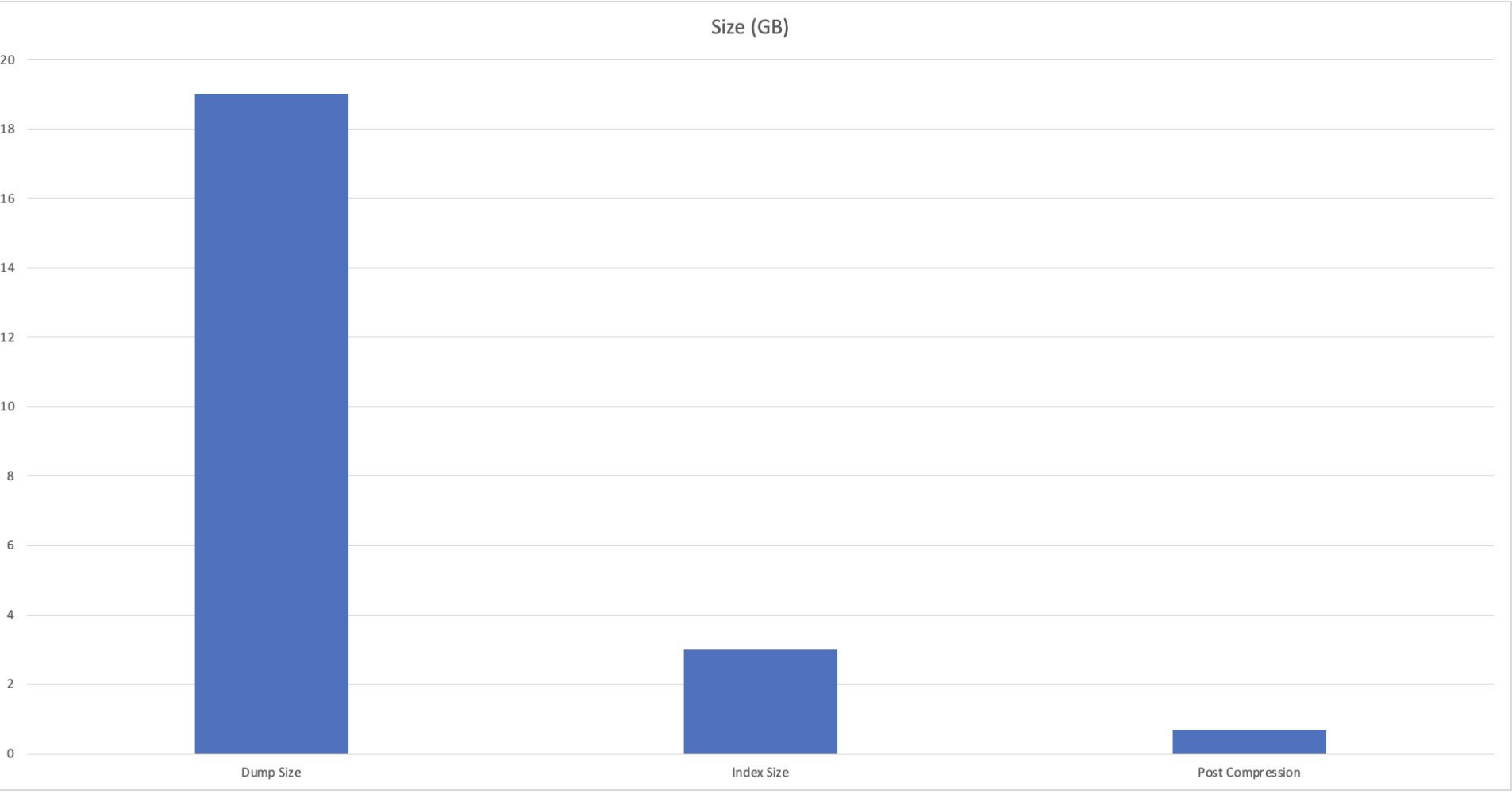

aws
Services
Search for services, features, marketplace products, and docs [Option+S]
rcs1
Global
Support
Amazon S3
We're continuing to improve the S3 console to make it faster and easier to use. If you have feedback on the updated experience, choose Provide feedback.
Provide feedback
Buckets
Access Points
Object Lambda Access Points
Batch Operations
Access analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
AWS Organizations settings
Feature spotlight 3
AWS Marketplace for S3
Amazon S3
Buckets (4)
Buckets are containers for data stored in S3. Learn more
Copy ARN
Empty
Delete
Create bucket
Find buckets by name
1
Name
AWS Region
Access
Creation date
adv-db
US East (Ohio) us-east-2
Bucket and objects not public
April 25, 2021, 20:47:35 (UTC-04:00)
index-files
US East (Ohio) us-east-2
Bucket and objects not public
April 27, 2021, 08:32:46 (UTC-04:00)
merged-index
US East (Ohio) us-east-2
Bucket and objects not public
April 27, 2021, 11:29:50 (UTC-04:00)
title-extract
US East (Ohio) us-east-2
Bucket and objects not public
April 27, 2021, 11:28:17 (UTC-04:00)

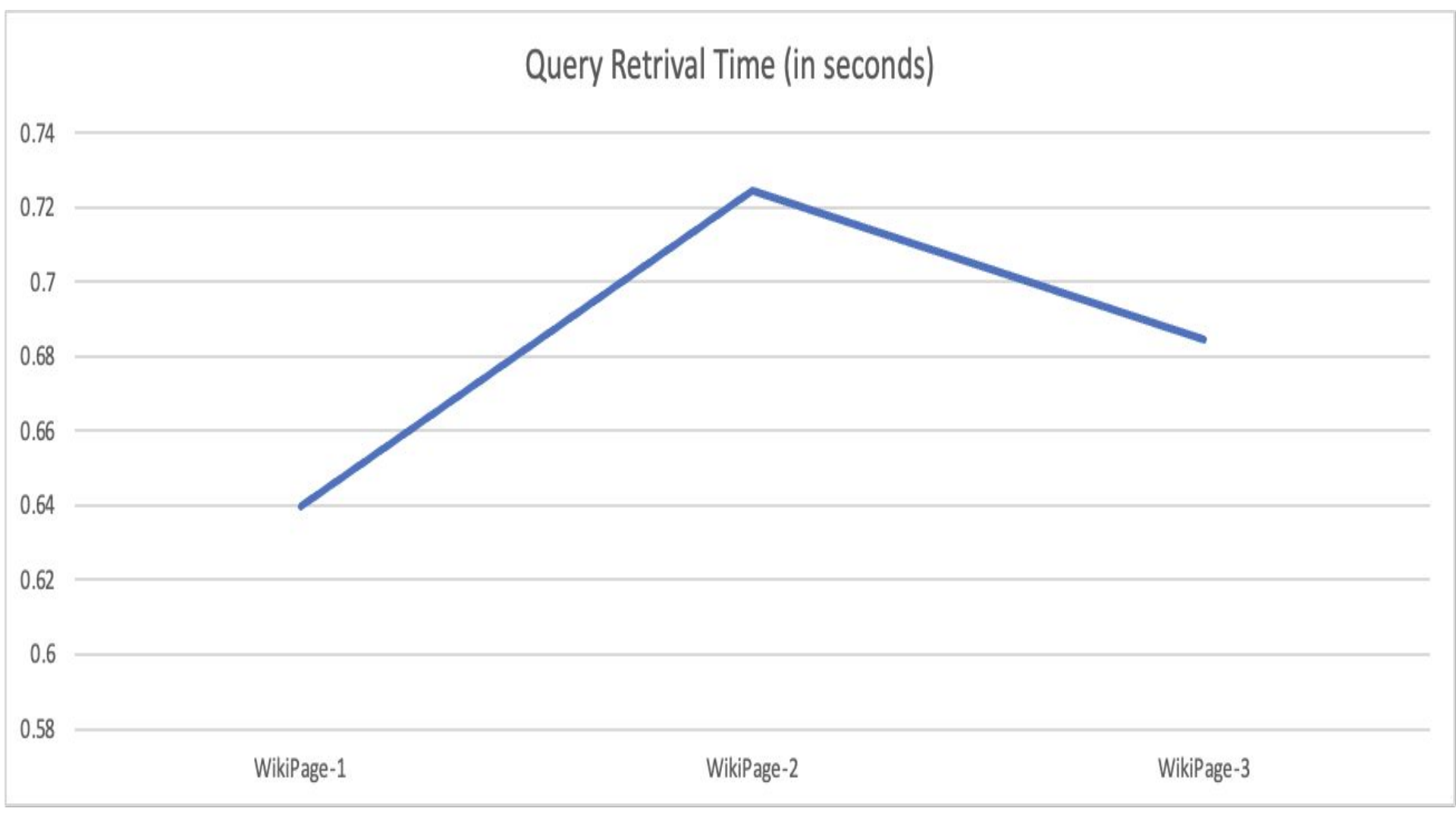
Why this Tech Stack

- Python - Most widely used programming language.
- Flask - lightweight
- S3 - Faster, Cheaper and easier to query
- ElasticSearch - preconfigured search cluster.

Evaluation plan

- Developing an application/system to showcase the search performance of a dataset over an inverted index
  - Search performance is the query retrieval time.
- Measuring the time taken to index large textual corpus. (Index creation time)
  - Overall time taken is 1.5 hours for complete dataset
- Inverted index quality: search result quality
  - Correct outputs
- Inverted Index size to be proportionally less than dump size.
  - Graph in next slide
- Scalability of the index
  - The index creation time for any number of articles is constant.


Size (GB)
20
18
16
14
12
10
8
6
4
2
0
Dump Size
Index Size
Post Compression


Query Retrival Time (in seconds)
0.74
0.72
0.7
0.68
0.66
0.64
0.62
0.6
0.58
WikiPage-1
WikiPage-2
WikiPage-3

DEMO

Contributions

Gaurav - Merging and Compression, Front End

Moulika - Data Preprocessing, S3 configuration

Rajath - Index Creation and Searching, Flask Backend linking

Future Scope:

Wikipedia History Page is a use-case of a version control system. It stores all the modification(versions) of a wikipedia page.

Creating a reliable inverted index for the history page is a difficult task due to the following reasons:

- Deleted content can have complications on the ranking of the wikipedia pages.
- Copyright infringement issues for a previous versions.

Conclusion:

- A framework for inverted index is useful for fielded and full-text search.
- Use of k-way merge provides an efficient merging sorted inverted index files.
- Perfect use case for a binary search.

Drawbacks:

- Ranking of wikipedia pages can depend on various other parameters and the list is exhaustive.

  Eg: PageRank algorithm ranks documents based on page links back to a webpage. An assumption is made that an arbitrary webpage is more relevant when it has more links pointing.

References:

- Yan, Hao, Shuai Ding, and Torsten Suel. “Inverted Index Compression and Query Processing with Optimized Document Ordering”. *Proceedings of the 18th International Conference on World Wide Web - WWW '09*. ACM Press, 2009.
- Zhong, Shaojun, Min Shang, and Zhijuan Deng. “A Design of the Inverted Index Based on Web Document Comprehending”. *Journal of Computers* 6.4 (2011).
- Dustin W. Kincaid Whitney S. Beck Jessica E. Brandt Margaret Mars Brisbin Kaitlin J. Farrell Kelly L. Hondula Erin I. Larson Arial J. Shogren. “Wikipedia can help resolve information inequality in the aquatic sciences”.

Thank You.

- Gaurav Jangir. (gjangir1@student.gsu.edu)
- Rajath CS (rcs1@student.gsu.edu)
- Moulika Achhe(machhe2@student.gsu.edu)