ہمکنار پاکستان
Urdu Search Engine

This document gives detailed description of developments that were made during the second milestone of this project. Key highlights are working prototypes of filtering, indexing and query response system. We have also developed a prototype for crawled data, which gave us inside knowledge for developing and managing our own cluster.

High Performance Computing and Networking Lab
Center for Language Engineering
Al-Khawarizmi Institute Of Computer Science,
University Of Engineering and Technology, Lahore

National ICT R&D Fund

Table of Contents

1. Introduction

In modern era, “search engine” has become the soul entity to provide desired information on just entering few relevant keywords. Our knowledge source is shifted from books and newspapers to web, mainly due to the fact that search engines give wide variety of relevant information in few seconds. Search engines can influence political views and change common perception, as users increasingly rely on them to navigate online content [1]. The search becomes more accurate and relevant when search engines are developed for a particular language, as majority of the languages are morphologically different than the others. Although, in classical language philosophy there is a divide among the linguists upon the treatment of the language (for example Chomsky’s vs. Quine’s) but each group acknowledges that there do exist unique language specific complexities that need to be catered. Unfortunately, the field of Natural Language Processing (NLP) is also less explored in the context of Urdu language. In the light of all this, a search engine that searches Urdu content against Urdu queries is not only challenging but also an exciting research area.

This project develops ‘Urdu Search Engine (USE)’ to address the national and linguistic needs, and to incubate the much needed expertise in this area of research and development. USE is a practical step to encourage not only research on Urdu but it also facilitates a large group of user communities who prefer to search and view information in Urdu. It is also an opportunity to investigate Urdu specific challenges in general. Text summarization is an active research area, and exploring it from the view of handling the complexities of Urdu language will give us an opportunity to contribute in the field besides providing a unique feature to a huge segment of the society. USE is developing content filtering which is according to our social and moral standards. For example, filtering out adult and inappropriate content, which is freely available on web. Modern search engines also overlook the requirements of low-end mobile users. A primary reason for this is that in the developed world the penetration of smart gadgets is overwhelming and the high-speed Internet services do not raise a need of developing convenient search mechanisms for the users of low-end mobile phones. This includes the development of an SMS-based search facility, which requires extracting a succinct summary of the search results and send it back to the user via SMS.

The project is working in three aspects, focusing on high performance distributed computing, content search optimization and local content management. Implementing in these three areas is a challenge in itself, as they are less explored by our local research community. Modern NLP applications perform computations over large corpora. With increasing frequency, NLP applications use the Web as their corpus and rely on queries to commercial search engines to support these computations [2] [3] [4] [5]. But search engines are designed and optimized to answer people’s queries, not as building blocks for NLP applications. As a result, the applications are forced to issue literally millions of queries to search engines, which can overload search engines, and limit both the speed and scalability of the applications. In response, Google has created the “Google API” to shunt programmatic queries away from Google.com and has placed hard quotas on the number of daily queries a program can issue to the API. Other search engines have also introduced mechanisms to block programmatic queries, forcing applications to introduce “courtesy waits” between queries and to limit the number of queries they issue. Having a “private” search engine would enable an NLP application to issue a much larger number of queries quickly [6].

2. Urdu Search Engine

Search engines are as critical to Internet use as any other part of the network infrastructure. Search engine is a program which searches the entire web and returns results (documents/ webpages) for a specific keyword or a query. Typically, search engine uses a program (crawler) to fetch as many documents as possible from the web. Another program unit called "indexer" is used to read these fetched documents and build an index based on the fetched documents word just like an index of a book. Each search engine algorithms use this index to search and return searched documents. They also create or use algorithms to return only meaningful results like for ranking or trending results, but their detailed internal workings are trade secret. Figure 1 shows the basic working of a search engine [7].

USE covers all the essential components of a sophisticated search engine like crawling, data indexing and search. Like any conventional search engine a web interface is provided to users for sending query to the system and obtaining the relevant results. It will also help out the mobile based users especially low end mobile users to send query to the systems using a number and receive information over the mobile phone. Urdu Search Engine is comprised of following components:

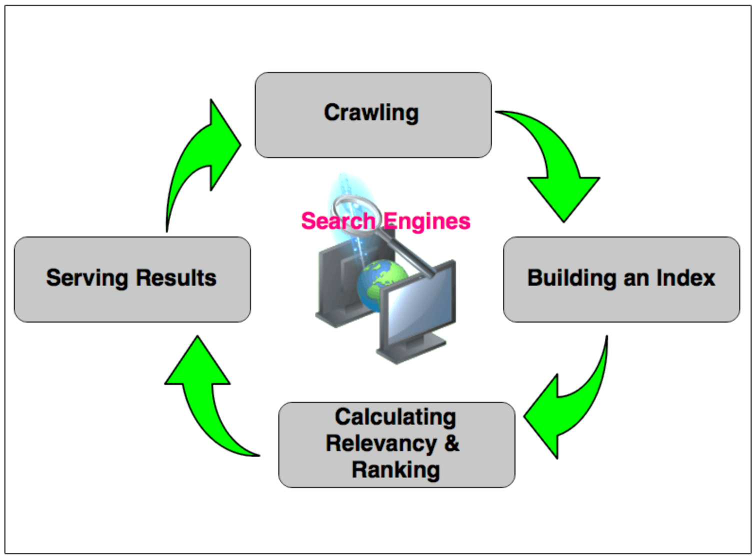


Figure 1: Generic infrastructure of a search engine

- **Cloud Infrastructure (CI):** CI deals with computing infrastructure, crawling web content for other teams, running and maintaining search software stack and to provide space for development, testing and deployment of the work. It provides a scalable storage and distributed computing infrastructure for USE. Incremental web crawling service to improve the seeds so the WWW crawling finishes in a desirable time is also included in the responsibilities of CI.
- **Information management (IM):** IM Component will be responsible to perform linguistic and textual analysis on raw content. IM will deal with policy management, language identification and content filtering by using well-defined machine learning algorithms to limit the search to Urdu contents only. It will also provide the implementation of tokenization, segmentation and text summarization of Urdu content to acquire the word boundaries.

- **Search management (SM):** SM will deal with collection of documents and building indexes which help out in processing the queries in order to provide satisfactory search results to the user. This component is responsible for Index prototyping and design to make the web pages useable for search. Ranking of the web pages depending upon the ranking policy and conversion of user query into meaningful piece of information by using Query Response system is part of SI.

Figure 2 shows the complete infrastructure of USE with color coded blocks. The green blocks shows the operations done by CI, grey are the component blocks of IM and pink is the block description of SM. The data that is crawled and managed by CI is passed on to MI for analysis and language processing. Whenever there is a query received by the SM is fetches indexed files, filters and rank the results according to polices and present to the user.

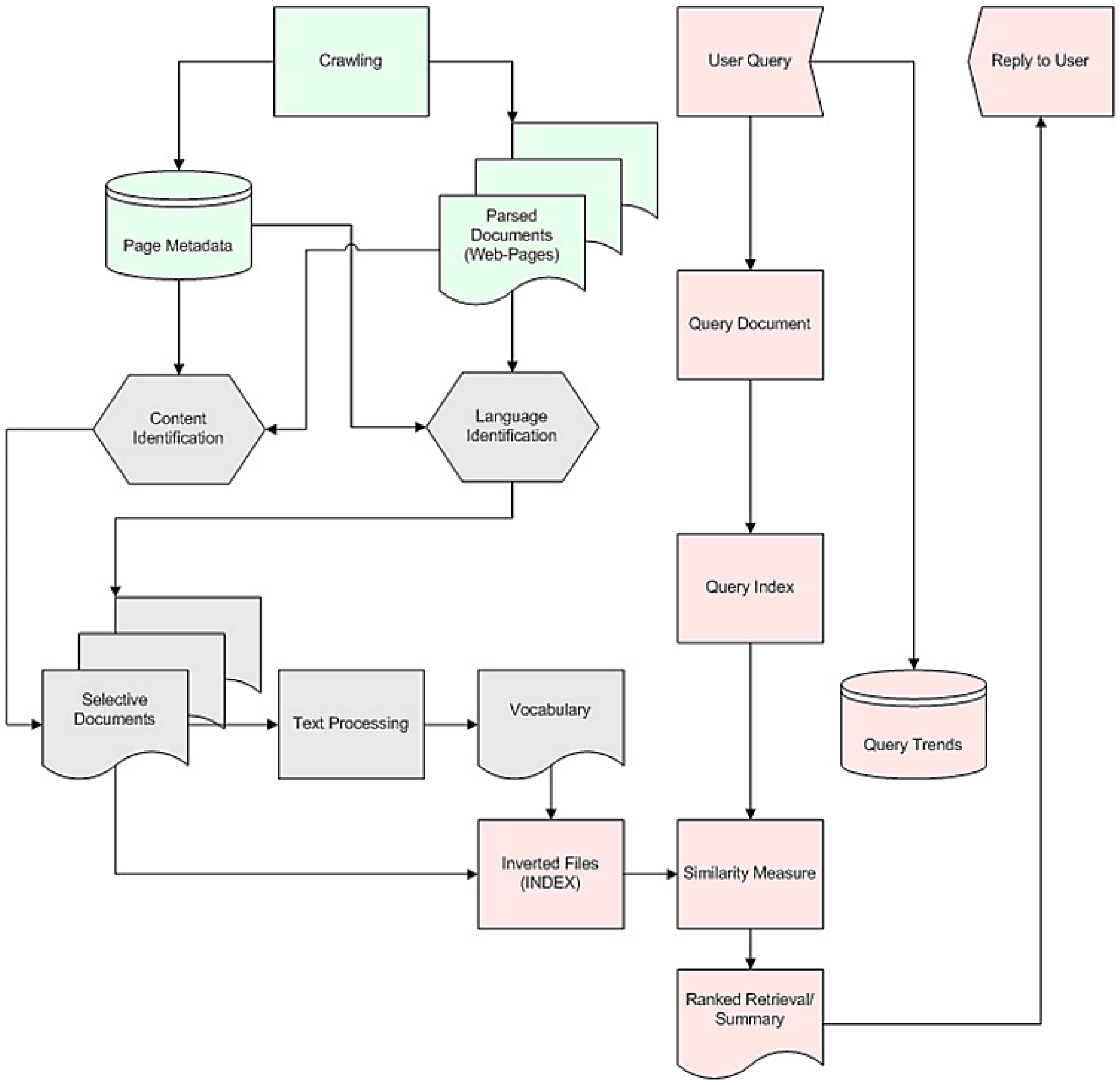


Figure 2: High level infrastructure of USE

3. Cloud Infrastructure

The computation infrastructure and storage requirements of this project are quite intense. Moreover, constant power and network failures are prevalent issues. Keeping in view all these limitations, a hybrid plan has been formed, which also minimizes the cost of the project. Instead of building entire CI infrastructure in-house, services of AWS were used. The required/extracted data was fetched out of AWS and stored in-house for use. This concurrent use of AWS cloud and local compute facility is referred as 'hybrid' approach. This hybrid plan is optimal in terms of cost. The first step CI did was to setup Amazon Web Services (AWS). We looked into the *Common Crawl* data available on AWS for the month of October 2016. This gave us a preliminary insight about how to crawl data and prototype it before making a full fledge cluster and maintaining it locally. For information storage a distributed file system is designed to hold a large amount of data and provide access to this data to many clients distributed across a network.

There are a number of distributed file systems that solve this problem in different ways. We have chosen Apache's 'Hadoop Distributed File System' (HDFS) as our primary source of information storage and retrieval. Few high speed 'Network Attached Storage' (NAS) boxes will complement our in-house storage needs.

3.1 Components

Further we briefly described the basic components used in our Implementation until now.

3.1.1 Infrastructure using Amazon Web Services (AWS)

Amazon Web Services (AWS) is a collection of remote computing services, which together make up a cloud computing environment. The service that is going to be predominantly utilized on this project is Amazon Elastic Compute Cloud (EC2). EC2 allows scalable deployment of applications by providing a web service through which a user can boot an 'Amazon Machine Image' to create a virtual machine, which Amazon calls an "instance", containing any software desired. A user can create, launch, and terminate server instances as needed, paying by the hour for active servers, hence the term "elastic". EC2 provides users with control over the geographical location of instances that allow latency optimization and high levels of redundancy. Amazon EC2 provides resizable compute capacity on demand. The processing, algorithms, crawling, content caching, corpus creation, model and index production, system maintenance, and end user interfaces can all be hosted on Amazon EC2.

Amazon EC2's simple web service interface allows users to obtain and configure capacity with minimal friction. It provides users with complete control of the computing resources and lets them run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing users to quickly scale capacity, both up and down, as their computing requirements change. Amazon EC2 changes the economics of computing by allowing users to pay only for capacity that they actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate themselves from common failure scenarios.

EC2 provides users with control over the geographical location of instances that allows for latency optimization and high levels of redundancy. For example, to minimize downtime, a user can set up server instances in multiple zones that are insulated from each other for most causes of failure such that one backs up the other. AWS is the pioneer in cloud computing IaaS model and the requirements of the project demand services similar to the ones offered by AWS. Therefore AWS was an obvious choice.

3.1.2 Information Storage Using Apache's HDFS

The HDFS is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. By distributing storage and computation across many servers, the resource can grow with demand economically at every size [8]. Following is a brief description of the HDFS system.

HDFS has master/slave architecture as shown in Figure 3. An HDFS cluster consists of a single *NameNode*, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of *DataNodes*, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of *DataNodes*. The *NameNode* executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to *DataNodes*. The *DataNodes* are responsible for serving read and write requests from the file system's clients. The *DataNodes* also perform block creation, deletion, and replication upon instruction from the Named Node. HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file

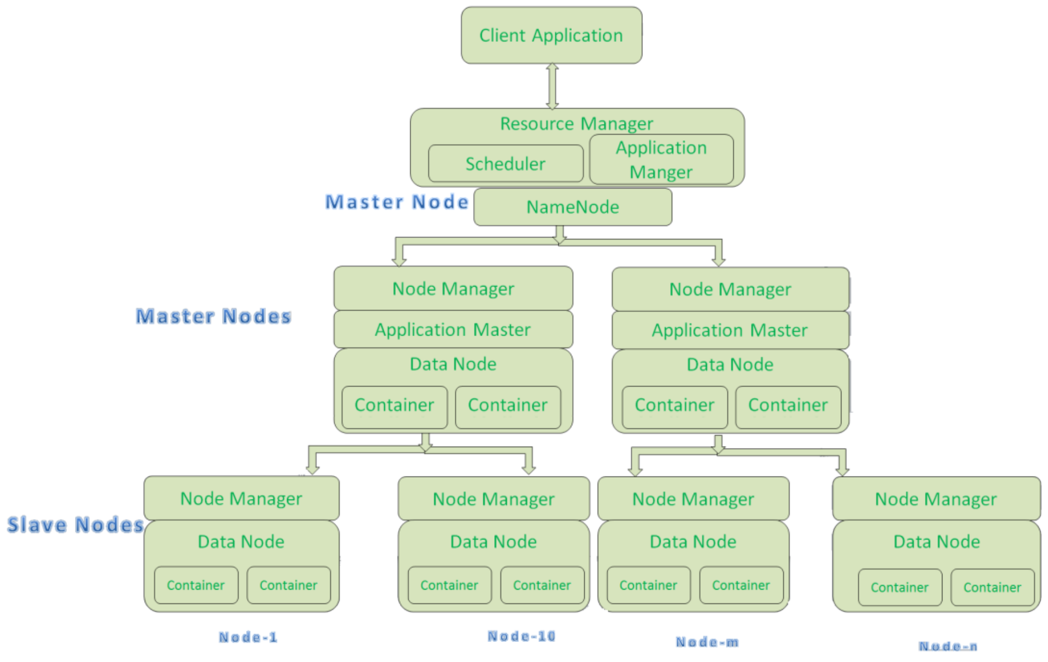


Figure 3: Internal architecture of Apache's HDFS

are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time. The *NameNode* makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a *Blockreport* from each of the *DataNodes* in the cluster. Receipt of a Heartbeat implies that the *DataNode* is functioning properly. A Block report contains a list of all blocks on a *DataNode* as depicted in Figure 4.

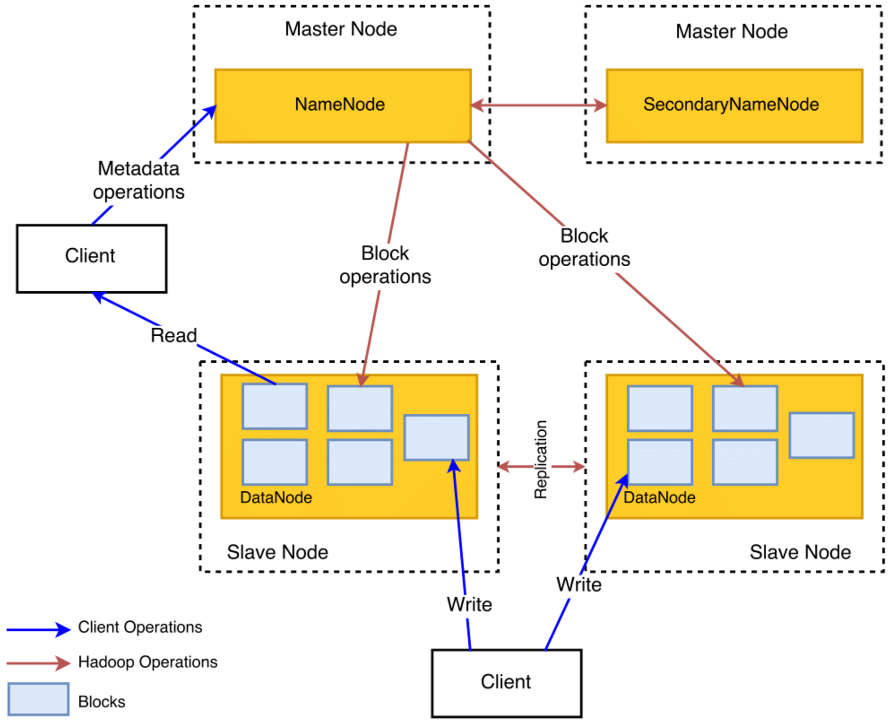


Figure 4: Detailed view of Hadoop Distributed File System

MapReduce

MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks. Users specify the computation in terms of a map and a reduce function, and the under-lying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks as shown in Figure 5. Programmers find the system easy to use more than ten thousand distinct MapReduce programs have been implemented internally at Google over the past four years, and an average of one hundred thousand MapReduce jobs are executed on Google's clusters every day, processing a total of more than twenty petabytes of data per day [9].

The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: map and reduce. Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the reduce function. The reduce function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges these values together to form a possibly smaller set of values. Typically just zero or one output value is produced per reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory.

Many different implementations of the MapReduce interface are possible. The right choice depends on the environment. For example, one implementation may be suitable for a small shared-memory machine, another for a large NUMA multiprocessor, and yet another for an even larger collection of networked machines. Several open source implementations of MapReduce have been developed [10] and the applicability of MapReduce to a variety of problem domains has been studied [11].

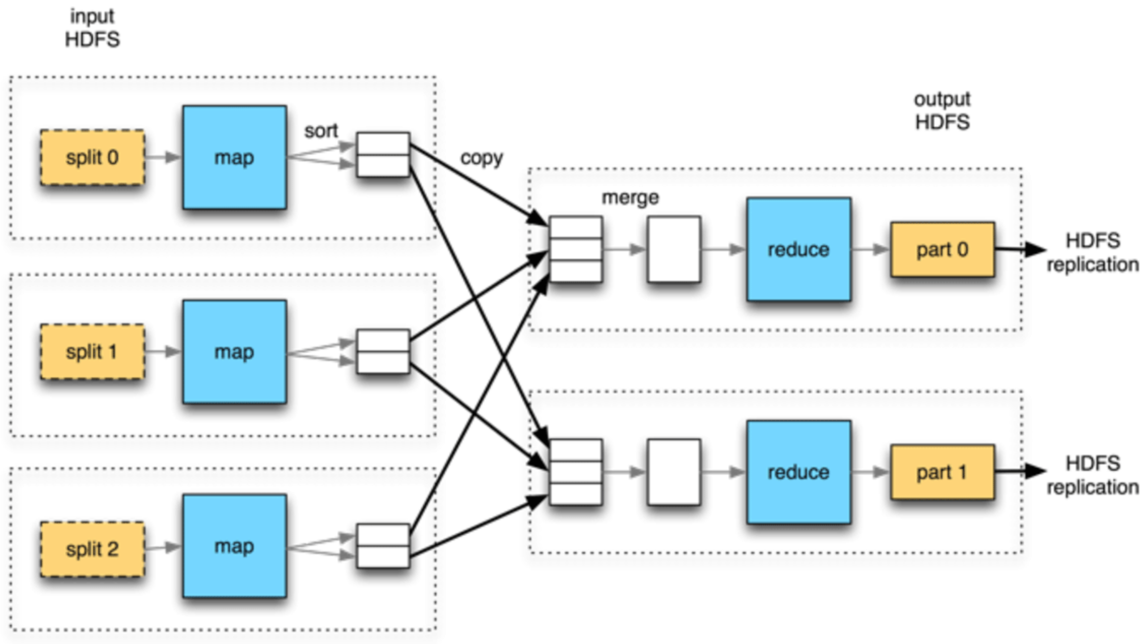


Figure 5: Execution overview using MapReduce

3.2 Implementation Methodology

In order to analyze the whole web, first step is crawling data from web pages. These web pages occupy a large volume of storage (more than 100 TB's). The analysis on this data requires a lot of computational power. The analysis filters out information about the contents of interest, in this case Urdu content is our main focus.

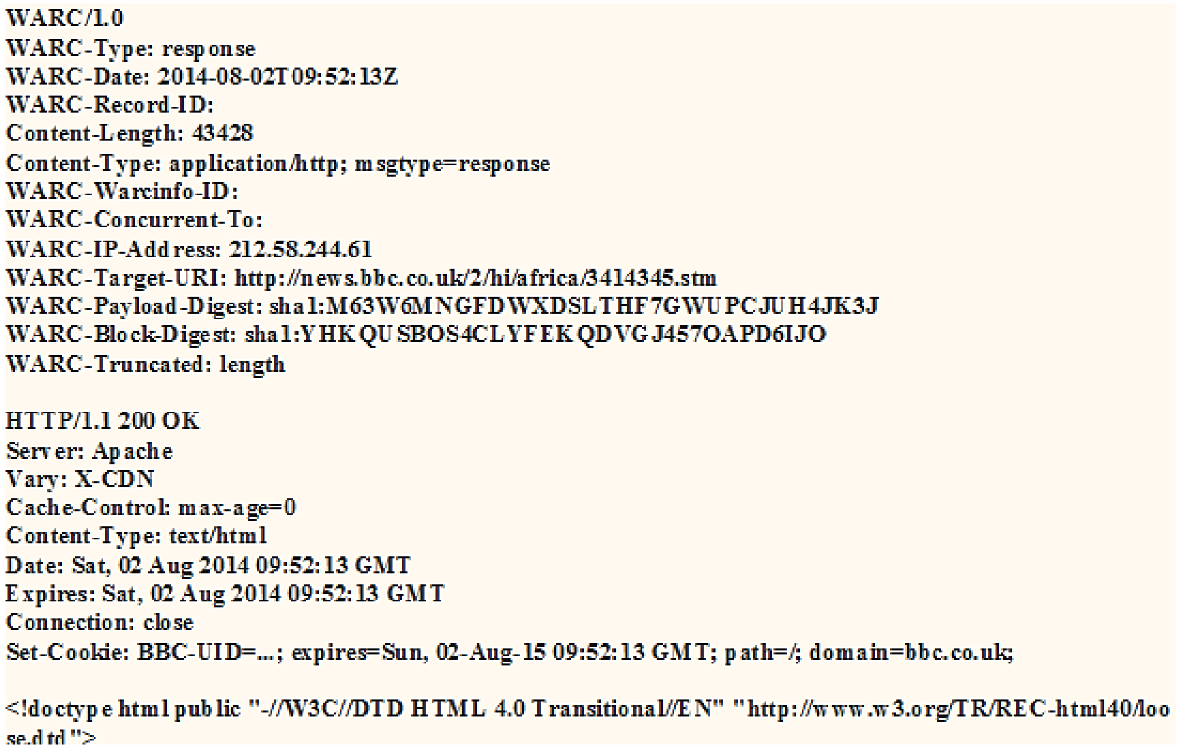
In order to analyze the web data, the entire web has to be crawled. Instead of "crawling" the whole web, an openly available crawled dataset provided by *Common Crawl* [12] could be used.

Common Crawl freely provides the crawled data every month. In order to analyze this data, a large number of machines are required. To make the task easy, Amazon provides web services for this purpose. Amazon has many online services comprising of virtual machines with desired operating systems, storage services and options for creating big clusters for computing big data.

3.2.1 Crawled Data

Common Crawl provides free crawled data every month. The data for October 2016 was used for the analysis. The data contains more than 3.25 million webpages [13]. Common crawl uses sitemaps to improve their seed. The data is provided in different formats. The details of the data are as follows:

- **WARC files:** These files contain crawler requests as well as the response headers along with the actual payload. WARC format is shown below. The snippet in figure 6 shows response header and actual payload sample. Response header has several fields of interest including IP address, URL, content length and content type [14].



```
WARC/1.0
WARC-Type: response
WARC-Date: 2014-08-02T09:52:13Z
WARC-Record-ID:
Content-Length: 43428
Content-Type: application/http; msgtype=response
WARC-Warcinfo-ID:
WARC-Concurrent-To:
WARC-IP-Address: 212.58.244.61
WARC-Target-URI: http://news.bbc.co.uk/2/hi/africa/3414345.stm
WARC-Payload-Digest: sha1:M63W6MNGFDWXDSLTHF7GWUPCJUH4JK3J
WARC-Block-Digest: sha1:YHKQUSBOS4CLYFEKQDVGJ457OAPD6IJO
WARC-Truncated: length

HTTP/1.1 200 OK
Server: Apache
Vary: X-CDN
Cache-Control: max-age=0
Content-Type: text/html
Date: Sat, 02 Aug 2014 09:52:13 GMT
Expires: Sat, 02 Aug 2014 09:52:13 GMT
Connection: close
Set-Cookie: BBC-UID=...; expires=Sun, 02-Aug-15 09:52:13 GMT; path=/; domain=bbc.co.uk;

<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/TR/REC-html40/loo
se.dtd">
```

Figure 6: Format of WARC file

- **WAT files:** WAT files store computed metadata for the data stored in WARC files. The metadata is computed for all three records in WARC file (request, response, and payload). The information is stored in WAT files as JSON is shown in figure 7.

```
>> data['Envelope']['WARC-Header-Metadata']['WARC-Type']
"response"
>> data['Envelope']['Payload-Metadata']['HTTP-Response-Metadata']['Headers']['Server']
"Apache"
>> data['Envelope']['Payload-Metadata']['HTTP-Response-Metadata']['HTML-Metadata']['Head']['Title']
" BBC NEWS | Africa | Namibia braces for Nujoma exit "
>> len(data['Envelope']['Payload-Metadata']['HTTP-Response-Metadata']['HTML-Metadata']['Links'])
42
>> data['Envelope']['Payload-Metadata']['HTTP-Response-Metadata']['HTML-Metadata']['Links'][28]
{"path": "A@/href", "title": "Home of BBC Sport on the internet", "url": "http://news.bbc.co.uk/sport1/hi/
default.stm"}
```

Figure 7: Format of WAT file

- **WET files:** WET files only store plain text (removing any html tags). Plain text of the html page is saved immediately after WARC headers as shown in figure 8.

```
WARC/1.0
WARC-Type: conversion
WARC-Target-URI: http://news.bbc.co.uk/2/hi/africa/3414345.stm
WARC-Date: 2014-08-02T09:52:13Z
WARC-Record-ID:
WARC-Refers-To:
WARC-Block-Digest: sha1:JROHLCS5SKMBR6XY46WXREW7RXM64EJC
Content-Type: text/plain
Content-Length: 6724

BBC NEWS | Africa | Namibia braces for Nujoma exit
...
President Sam Nujoma works in very pleasant surroundings in the small but beautiful old State House...
```

Figure 8: Format of WET file

The use of WARC file is convenient as the file contained all the information required to carry out an in depth analysis. The file includes whole webpage along with all html tags, which is necessary to separate various parameters during analysis. Common Crawl corpus for October 2016 has over 100TB of data. To perform operations on this large volume of information requires a well-established infrastructure.

3.2.2 Elastic Map Reduce (EMR)

To perform operations on big data amazon provides EMR (Elastic Map Reduce) service. EMR service can be accessed from amazon web page [15]. Common crawl data from October 2016 dataset was used for analysis. The dataset is available on amazon public dataset which can be accessed using amazon S3 protocol for free [16]. As the data being processed is huge, the EMR job has to process large chunks of data. In order to avoid any memory related issues, Hadoop was configured on EMR. The output from Hadoop job was stored in S3 buckets. S3 is amazon web storage service.

Amazon EMR job runs on a cluster of EC2 instances. Each instance is a virtual machine which is configurable according to requirements. Amazon provides S3 storage for storing any input or output data. An EMR job running on EC2 clusters may take quite a long time to complete. The usage of resources can be monitored by Amazon Cloud Watch service. The flow of job is shown in the figure 9 [17].

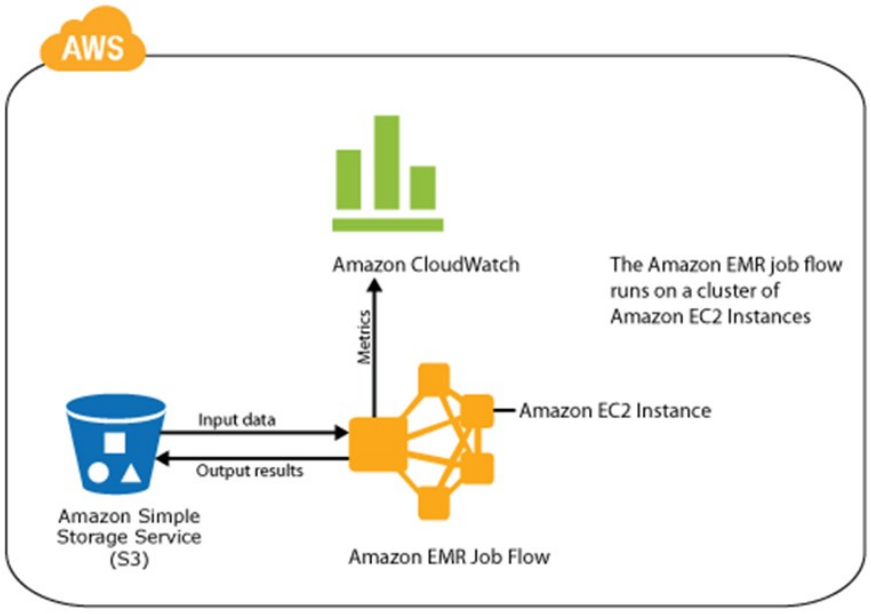

AWS
Amazon CloudWatch
The Amazon EMR job flow runs on a cluster of Amazon EC2 Instances
Metrics
Amazon EC2 Instance
Input data
Output results
Amazon Simple Storage Service (S3)
Amazon EMR Job Flow

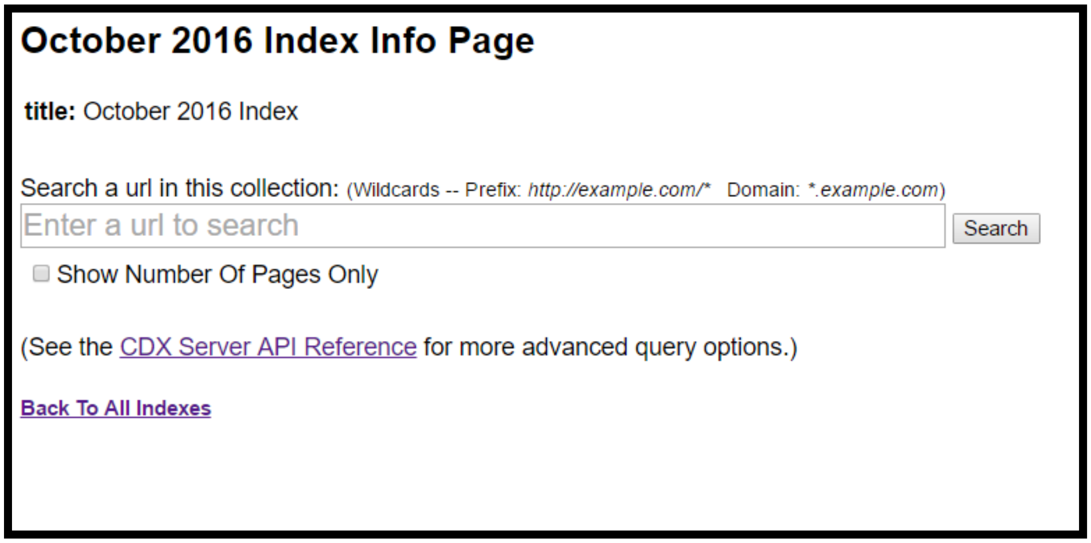
Figure 9: Shows the basic mechanism how EMR Job runs on AWS

3.2.3 Extraction of .pk domains

The strategy to obtain Urdu content involved extraction of .pk domain based web pages and analyzing the pages for Urdu content. The outlines of this strategy are as follows.

- Obtain all .pk based websites from common crawl dataset by using Map Reduce strategy.
- Obtain general information about these webpages, for example number of pages, hosting providers, document types and their numbers etc.
- Analyze these webpages for Urdu content using a language detection tool.
- Separate the pages rich in Urdu content.

All the steps above are mandatory to obtain the required results, however the list of webpages with *.pk* domains and their data could be found by common crawl URL index. The index provides the name of the WARC file where a website's data may be stored. It also provides the total length of that data along with the offset, representing the starting point of the data. This way, the need to write custom scripts for isolating *.pk* web pages from rest of the web could be avoided. The interface of common crawl URL index is shown in the figure 10. A list of all .pk webpages and corresponding WARC files can be obtained using this tool. The figure 11 shows the input query to obtain all domain names containing .pk. A portion of the list of *.pk* webpages after executing this query is shown in figure 12. The files are accessed and the required content is fetched using offset and length of that content using the python script, which can found in *code* attached.

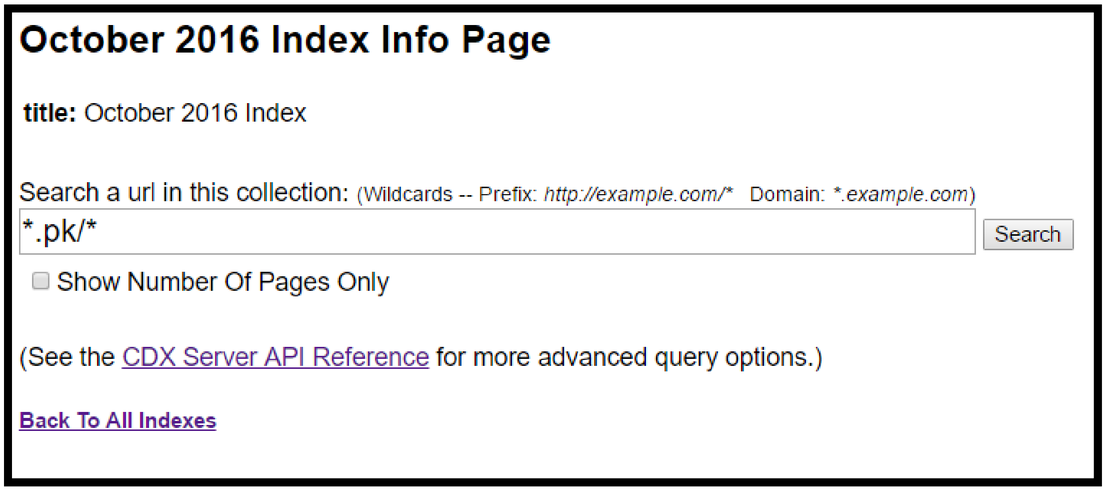

October 2016 Index Info Page

title: October 2016 Index

Search a url in this collection: (Wildcards -- Prefix: http://example.com/* Domain: *.example.com)

(See the CDX Server API Reference for more advanced query options.)

Figure 10: Webpage displaying interface for using Common Crawl Index


October 2016 Index Info Page

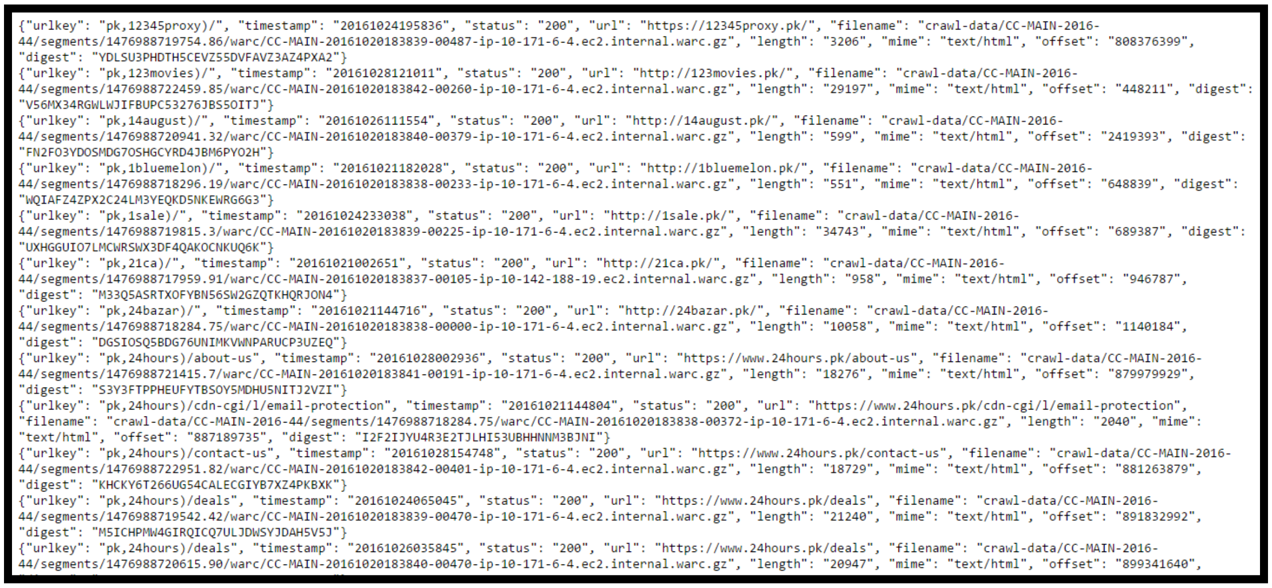
title: October 2016 Index

Search a url in this collection: (Wildcards -- Prefix: http://example.com/* Domain: *.example.com)

.pk/

(See the CDX Server API Reference for more advanced query options.)

Figure 11: Query to obtain list of .pk domains stored in Common Crawl data for October 2016.



```
{"urlkey": "pk,12345proxy)/", "timestamp": "20161024195836", "status": "200", "url": "https://12345proxy.pk/", "filename": "crawl-data/CC-MAIN-2016-
44/segments/1476988719754.86/warc/CC-MAIN-20161020183839-00487-ip-10-171-6-4.ec2.internal.warc.gz", "length": "3206", "mime": "text/html", "offset": "808376399",
"digest": "YDLSU3PHDTH5CEVZ55DVFAVZ3AZ4PXA2"}
{"urlkey": "pk,123movies)/", "timestamp": "20161028121011", "status": "200", "url": "http://123movies.pk/", "filename": "crawl-data/CC-MAIN-2016-
44/segments/1476988722459.85/warc/CC-MAIN-20161020183842-00260-ip-10-171-6-4.ec2.internal.warc.gz", "length": "29197", "mime": "text/html", "offset": "448211", "digest":
"V56MX34RGWLWJIFBUPC53276JBS5OITJ"}
{"urlkey": "pk,14august)/", "timestamp": "20161026111554", "status": "200", "url": "http://14august.pk/", "filename": "crawl-data/CC-MAIN-2016-
44/segments/1476988720941.32/warc/CC-MAIN-20161020183840-00379-ip-10-171-6-4.ec2.internal.warc.gz", "length": "599", "mime": "text/html", "offset": "2419393", "digest":
"FN2FO3YDOSMDG7OSHGCYRD4JBM6PYO2H"}
{"urlkey": "pk,1bluemelon)/", "timestamp": "20161021182028", "status": "200", "url": "http://1bluemelon.pk/", "filename": "crawl-data/CC-MAIN-2016-
44/segments/1476988718296.19/warc/CC-MAIN-20161020183838-00233-ip-10-171-6-4.ec2.internal.warc.gz", "length": "551", "mime": "text/html", "offset": "648839", "digest":
"WQIAFZ4ZPX2C24LM3YEQKD5NKEWRG6G3"}
{"urlkey": "pk,1sale)/", "timestamp": "20161024233038", "status": "200", "url": "http://1sale.pk/", "filename": "crawl-data/CC-MAIN-2016-
44/segments/1476988719815.3/warc/CC-MAIN-20161020183839-00225-ip-10-171-6-4.ec2.internal.warc.gz", "length": "34743", "mime": "text/html", "offset": "689387", "digest":
"UXHGGUIO7LMCWRSWX3DF4QAKOCNKUQ6K"}
{"urlkey": "pk,21ca)/", "timestamp": "20161021002651", "status": "200", "url": "http://21ca.pk/", "filename": "crawl-data/CC-MAIN-2016-
44/segments/1476988717959.91/warc/CC-MAIN-20161020183837-00105-ip-10-142-188-19.ec2.internal.warc.gz", "length": "958", "mime": "text/html", "offset": "946787",
"digest": "M33Q5ASRTXOFYBN56SW2GZQTKHQRJON4"}
{"urlkey": "pk,24bazar)/", "timestamp": "20161021144716", "status": "200", "url": "http://24bazar.pk/", "filename": "crawl-data/CC-MAIN-2016-
44/segments/1476988718284.75/warc/CC-MAIN-20161020183838-00000-ip-10-171-6-4.ec2.internal.warc.gz", "length": "10058", "mime": "text/html", "offset": "1140184",
"digest": "DGSIOSQ5BDG76UNIMKVWNPARUCP3UZEQ"}
{"urlkey": "pk,24hours)/about-us", "timestamp": "20161028002936", "status": "200", "url": "https://www.24hours.pk/about-us", "filename": "crawl-data/CC-MAIN-2016-
44/segments/1476988721415.7/warc/CC-MAIN-20161020183841-00191-ip-10-171-6-4.ec2.internal.warc.gz", "length": "18276", "mime": "text/html", "offset": "879979929",
"digest": "S3Y3FTPPHEUFYTBSOY5MDHU5NITJ2VZI"}
{"urlkey": "pk,24hours)/cdn-cgi/l/email-protection", "timestamp": "20161021144804", "status": "200", "url": "https://www.24hours.pk/cdn-cgi/l/email-protection",
"filename": "crawl-data/CC-MAIN-2016-44/segments/1476988718284.75/warc/CC-MAIN-20161020183838-00372-ip-10-171-6-4.ec2.internal.warc.gz", "length": "2040", "mime":
"text/html", "offset": "887189735", "digest": "I2F2IJYU4R3E2TJLHI53UBHHNNM3BJNI"}
{"urlkey": "pk,24hours)/contact-us", "timestamp": "20161028154748", "status": "200", "url": "https://www.24hours.pk/contact-us", "filename": "crawl-data/CC-MAIN-2016-
44/segments/1476988722951.82/warc/CC-MAIN-20161020183842-00401-ip-10-171-6-4.ec2.internal.warc.gz", "length": "18729", "mime": "text/html", "offset": "881263879",
"digest": "KHCKY6T266UG54CALECGIYB7XZ4PKBXK"}
{"urlkey": "pk,24hours)/deals", "timestamp": "20161024065045", "status": "200", "url": "https://www.24hours.pk/deals", "filename": "crawl-data/CC-MAIN-2016-
44/segments/1476988719542.42/warc/CC-MAIN-20161020183839-00470-ip-10-171-6-4.ec2.internal.warc.gz", "length": "21240", "mime": "text/html", "offset": "891832992",
"digest": "M5ICHPMW4GIRQICQ7ULJDWSYJDAH5V5J"}
{"urlkey": "pk,24hours)/deals", "timestamp": "20161026035845", "status": "200", "url": "https://www.24hours.pk/deals", "filename": "crawl-data/CC-MAIN-2016-
44/segments/1476988720615.90/warc/CC-MAIN-20161020183840-00470-ip-10-171-6-4.ec2.internal.warc.gz", "length": "20947", "mime": "text/html", "offset": "899341640",
```

Figure 12: A list of .pk domains obtained after executing the query.

3.3 Results

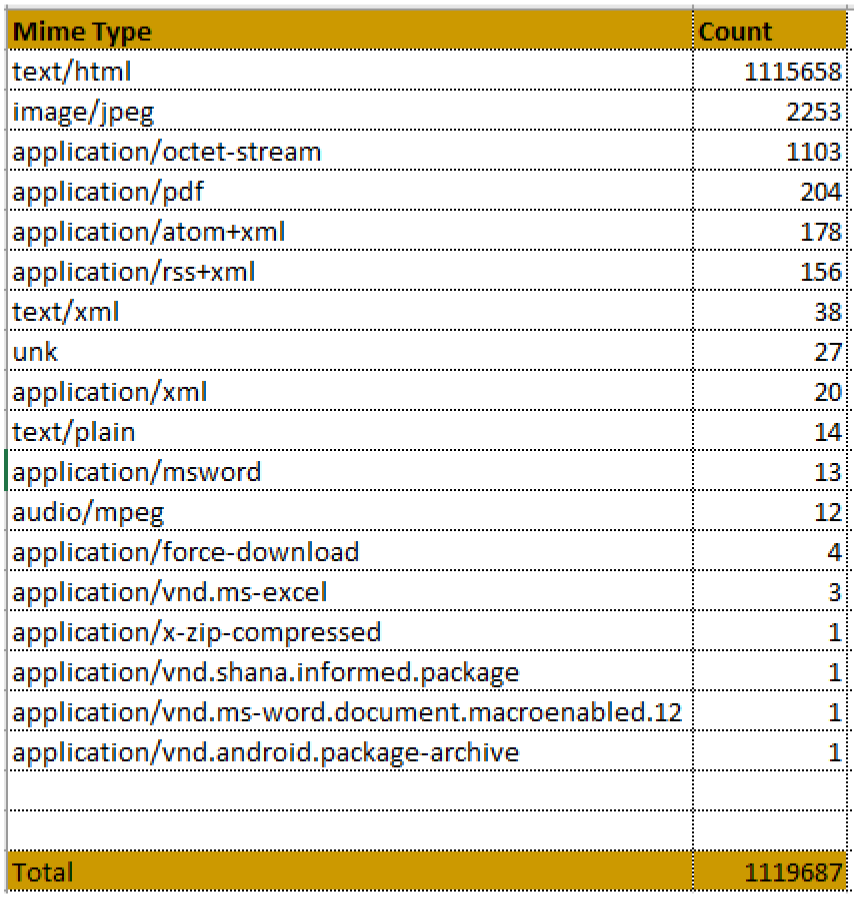
The total number of *.pk* webpages found, using the method above were almost 1.2 million. About 0.6 million web pages were selected for our analysis. A series of operations were performed on the data obtained from common crawl corpus. Results corresponding to various factors were derived and analyzed further.

The analysis showed that only 0.01% of the web is occupied by *.pk* domains. Most of the *.pk* usage is dominated by *.com* and *.edu* second level domains. The results show that html images and pdf are most abundant content types on the *.pk* pages. Most of the *.pk* pages are hosted on US servers and most of the content hosted on them is English.

3.3.1 MIME Type Vs Frequency

Table 1 is showing the types of document and their frequencies. The result shows that among 1119687 analyzed webpages, 1115658 were html content which dominates the document type overall. Images are second most used content among .pk domains. The reason is evident, as most of the content in a website is html, it has to be on top, images and PDFs are also abundant in number.

Table 1: Displaying document type along with number of its occurrences in 0.6 million documents of .pk domains.



Mime Type	Count
text/html	1115658
image/jpeg	2253
application/octet-stream	1103
application/pdf	204
application/atom+xml	178
application/rss+xml	156
text/xml	38
unk	27
application/xml	20
text/plain	14
application/msword	13
audio/mpeg	12
application/force-download	4
application/vnd.ms-excel	3
application/x-zip-compressed	1
application/vnd.shana.informed.package	1
application/vnd.ms-word.document.macroenabled.12	1
application/vnd.android.package-archive	1
Total	1119687

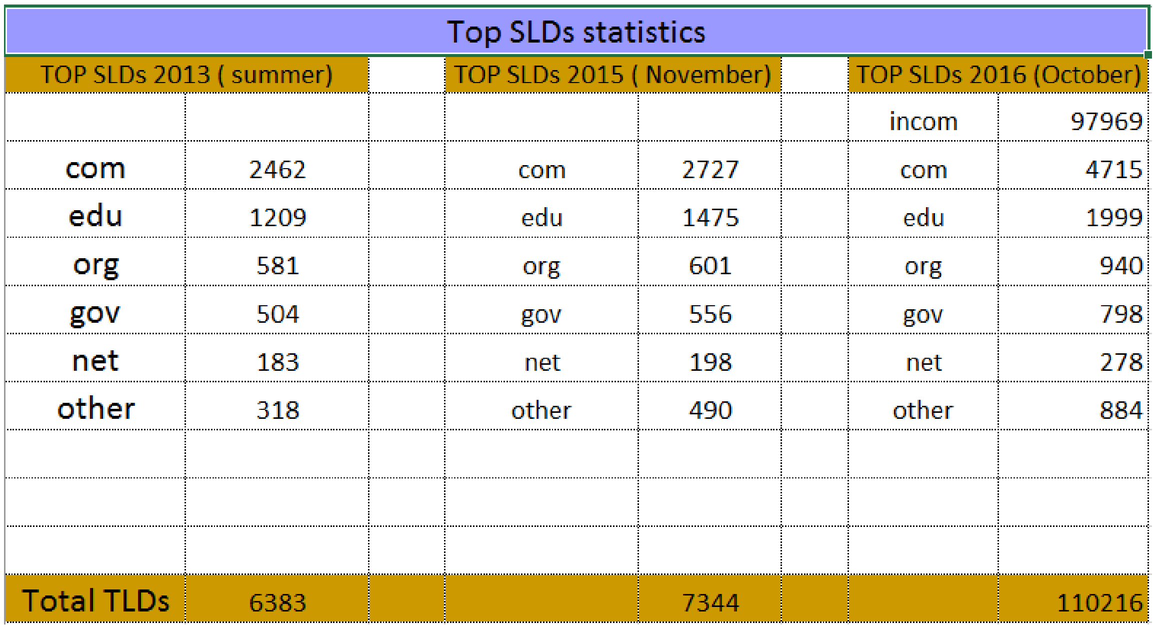
3.3.2 *Top Second Level Domains*

Among .pk webpages, a number of second level domains were found. A list of the domains and their number is given below. Table 2 shows top second level domains for 2013, 2015 and 2016 for .pk domain. The total number of domains seems to grow which is consistent with the fact that every year number of internet users and websites grow. However, in 2016 data, .incom dominated .com by quite a big margin. The reason is .incom is a free hosting service which provides domain names with its second level domain. Many Pakistani users in 2016 use this free service to host their website. Excluding this second level domain leaves .com on top and .edu as runner up.

3.3.3 *Top Hosting Countries*

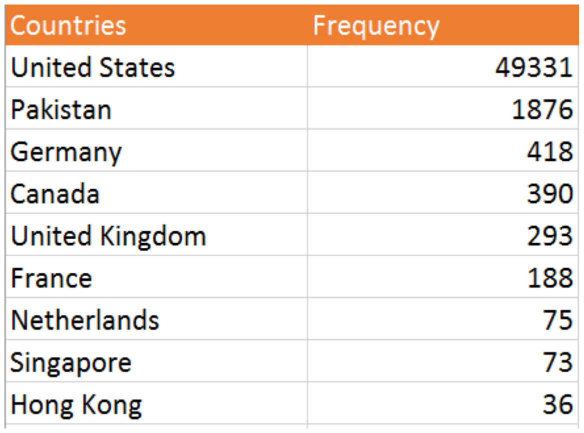
The header segment of each entry in the WARC file contains an IP address field. Based on the IP address the location of the server serving the page can be determined. MaxMind [18] provides IP to Geo resolving services. MaxMind provides databases for location look up. GeoLite is MaxMind's free set of databases for location look up. Using MaxMind's databases revealed following information. Table 3 shows the most used servers for hosting *.pk* websites are those of US. The reason for US being on top and not Pakistan may be the fact that US web hosting companies provide better interface and cost effective solutions. Thus the users tend to use easy solutions.

Table 2: Comparison of information regarding second level domains from common crawl data for 2013, 2015, 2016.



Top SLDs statistics							
TOP SLDs 2013 (summer)			TOP SLDs 2015 (November)			TOP SLDs 2016 (October)	
						incom	97969
com	2462		com	2727		com	4715
edu	1209		edu	1475		edu	1999
org	581		org	601		org	940
gov	504		gov	556		gov	798
net	183		net	198		net	278
other	318		other	490		other	884
Total TLDs	6383			7344			110216

Table 3: Number of documents arranged with respect to Country. The table depicts US based servers to host most of the *.pk* pages than Pakistan based servers.



Countries	Frequency
United States	49331
Pakistan	1876
Germany	418
Canada	390
United Kingdom	293
France	188
Netherlands	75
Singapore	73
Hong Kong	36

4. Information Management

The collection of raw content retrieved after crawling is ineffective without linguistic and text analysis. To use this content as meaningful information there is need to perform language processing on the content that includes language & content identification, segmentation of the content into words and conversion of the words into their base forms. Figure 13 shows the structure of a preprocessor that will be responsible for converting webpage content into a form that can be processed by individual filters. The Normalizer will strip the input of all punctuation, digits, diacritics, and special characters. The "Content Segmenter" will then split the webpage into sentences, and the Sentence Tokenizer will split sentences into individual words. Finally, the "Stop word Remover" may be used as required to eliminate Urdu "stopwords" from the input.

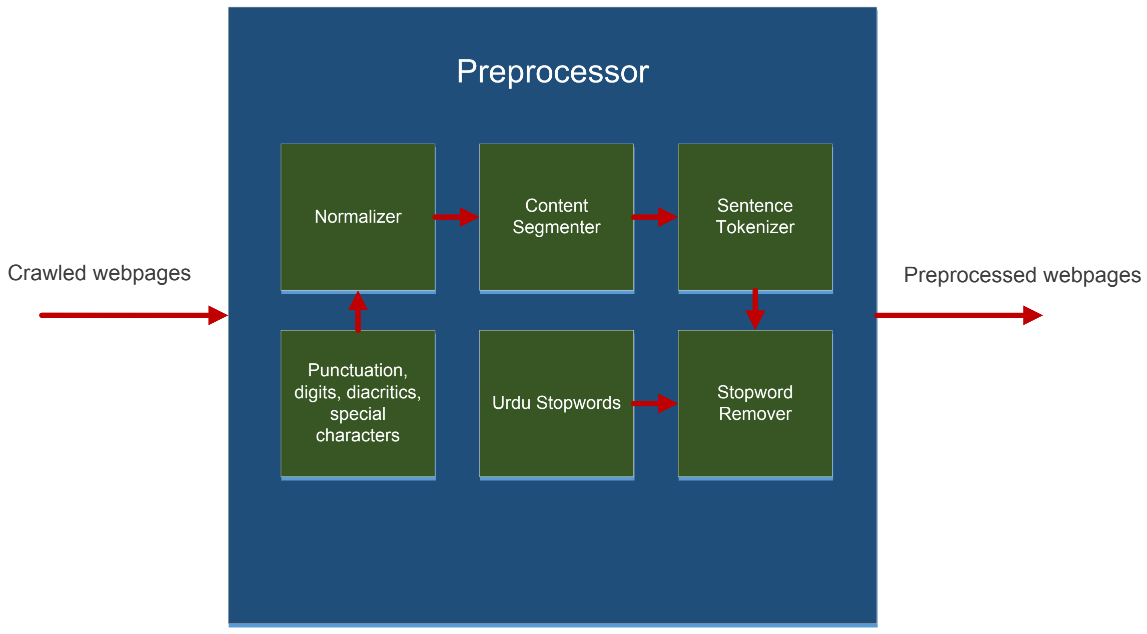


Figure 13: Internals of Preprocessor.

4.1 Architecture of Filtering System

As first step we implemented a prototype of working filtering system. The filtering system of the Urdu Search Engine is responsible for calculating a number of heuristics from webpages in order to decide whether to index them or not, or how to rank them when returning search results. This information would allow a better user experience by ensuring that emphasis is laid on relevant, up-to-date, and appropriate material.

The overall architecture of the filtering system is given in Figure 14. The filtering system is itself composed of four subsystems, each of which extracts a certain kind of information from crawled webpages. Our focus is on a webpage's language, content, age, and size. The working of each of these filters is described in detail in the sections that follow.

Additionally, each of these filters will be composed of two separate components: the policy and the filter itself. The filter will be responsible for processing a webpage, extracting the required information from it, and representing this information on a continuous scale from 0 to 1. This scale may take up different meanings depending on the filter in question: for language, it will represent the proportion of Urdu in a webpage, while for content; it will represent the severity of profanity in a webpage. In the age and size filters, this scale will represent a mapping from the age and size onto the webpage's significance, respectively.

The policy component of the filters will be responsible for setting a threshold between the extremes of 0 and 1 which webpages must satisfy in each of the categories in order to make into the search engine index or rank highly in users' search results. One might set the policy to allow only webpages with a certain amount of Urdu and under a certain threshold for profanity. An age threshold may be used to rank more recent, 'fresh' webpages higher than older matches for the same query. Similarly, one may use the size filter to push light-weight, faster-loading webpages higher up in search results when the engine is queried from a mobile device (since mobile data can be expensive). The relationship between the two components of each filter is shown in the figure 14, wherein the policy defines the thresholds that a given webpage must satisfy in order for the filter to let it through.

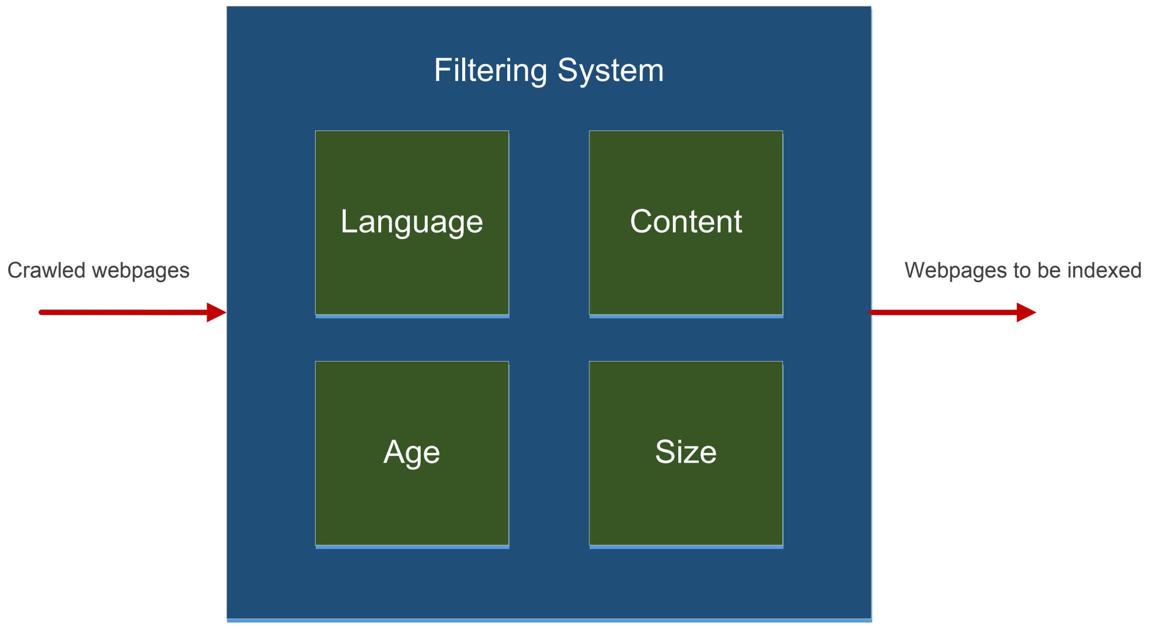


Figure 14 System Architecture

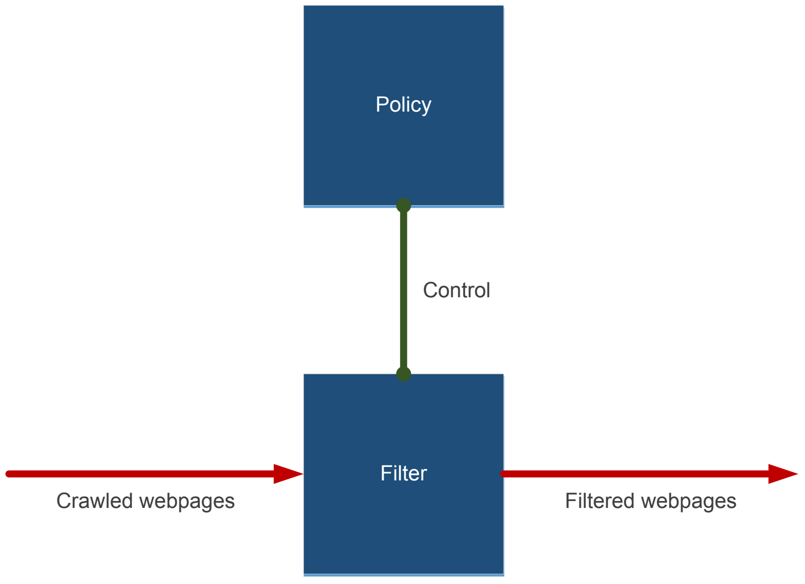


Figure 15: Policy

4.2 Language

The language filter is responsible for determining whether a particular webpage contains Urdu, and if it does, estimate its proportion in it. This is important since the Urdu Search Engine is primarily focused on crawling and indexing webpages with Urdu content and when doing open-ended crawling, it is important to know which webpages to keep and which to discard. A language filter will be able to sift through crawled webpages in order to keep for indexing only those that contain a substantial proportion of Urdu in them.

The design of the language filter is outlined in Figure 15. The Preprocessor cleans, normalizes, and tokenizes the webpage's content into an array of words. This is to decompose a single string containing the entire content into individual words we can then classify as Urdu or non-Urdu. The Preprocessor's output is passed to an Urdu Verifier which is responsible for determining whether there exists any Urdu text in the webpage at all.

This is done in two steps. First, the content is checked for the existence of any Arabic Unicode characters. Since all Urdu characters are a subset of the Arabic Unicode character set, any webpage that contains Urdu text must contain characters from within this range. If no Arabic Unicode characters are found, the webpage can be dropped right there and then. If, however, it does contain Arabic Unicode characters, then it must be checked for the presence of Urdu in particular, since Arabic Unicode characters are used for writing numerous other languages as well. This check is performed by searching the webpage content for any Urdu stopwords. A language's stopwords are those words that are especially common in its discourse. Our system matches the input against a precompiled list of Urdu stopwords. If any are found in the webpage, then it can be looked at in more detail to estimate the proportion of Urdu in it.

The Segment Extractor and Proportion Estimator make up the final parts of the language filter. They use a language model trained on a large Urdu corpus in order to classify Urdu words. Once all words have been classified, the proportion of Urdu in the webpage is then calculated and returned as a value in the range 0 to 1. This value is then checked against the policy thresholds in order to determine if the webpage in question should make it through the filter or not.

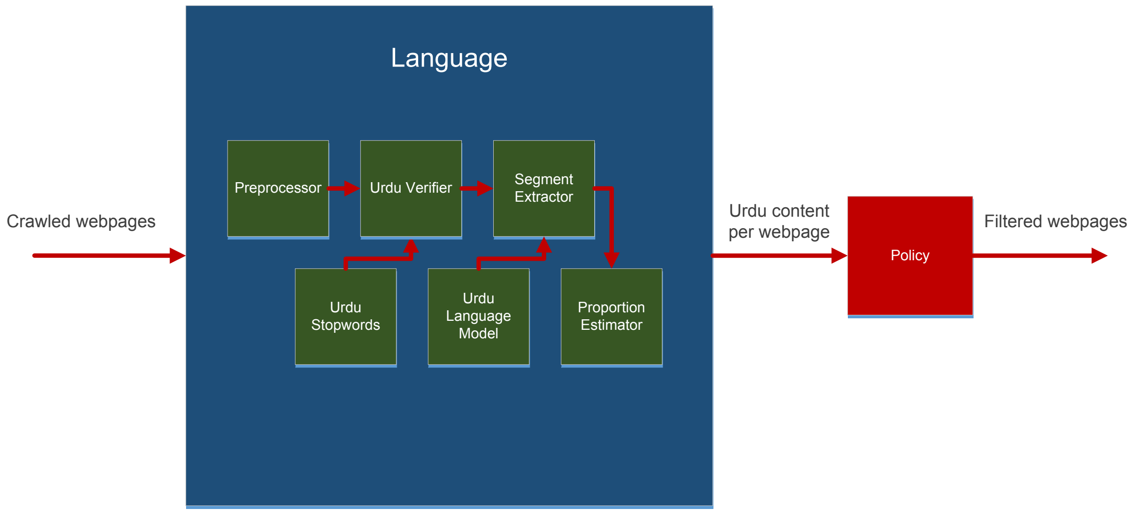


Figure 16 Language

4.3 Content

Content filtering is an important part of the crawling and indexing process since we do not wish to index and make available to users content whose language is deemed inappropriate or offensive. After the Preprocessor has cleaned, normalized and tokenized the content into words, a Profane Content Marker will look for any obscene words among them. This will be done using a precompiled profanity dictionary. The dictionary will contain a list of common Urdu swear words and phrases, and their respective severity scores on a graded scale. If no matches are found, then the content is deemed unobjectionable and is passed through the filter. If, however, profanity is found in the content, then it must be evaluated in order to determine its severity. This is important since not all swear words or phrases are equally bad. Additionally, combinations of supposedly mild profanity can yield much stronger phrases. A Sentence Profanity Accumulator will be used in this case to measure this compound effect over a window of words. The output of this Accumulator will be used by the Document Profanity Scorer to assign the webpage an overall score taking into the account the frequency, severity, and proximity of any profanity found. This overall score must then satisfy the thresholds set by the policy in order to pass through the filter.

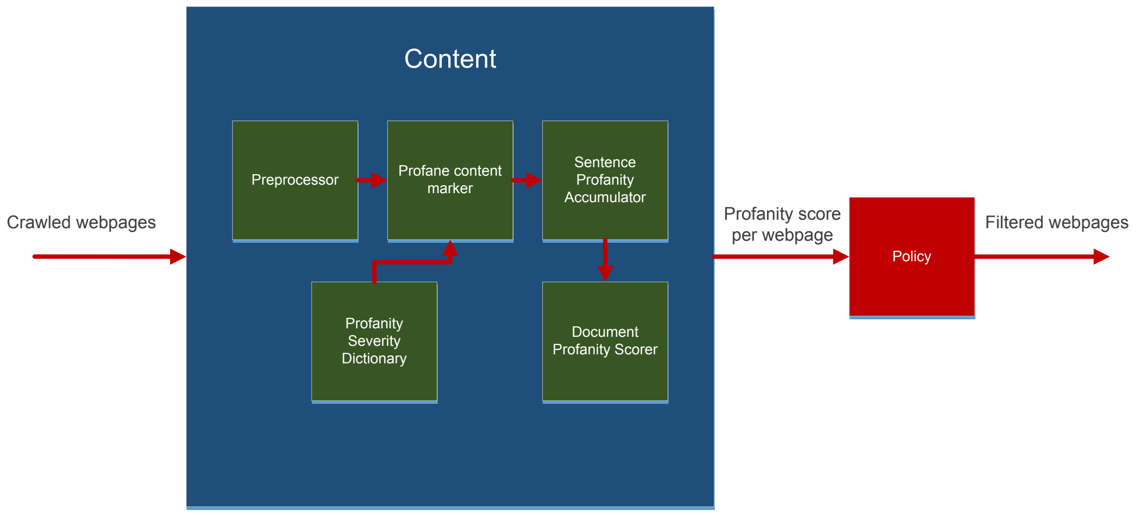

Content
Preprocessor
Profane content marker
Sentence Profanity Accumulator
Crawled webpages
Profanity score per webpage
Policy
Filtered webpages
Profanity Severity Dictionary
Document Profanity Scorer

Figure 17: Content

4.4 Age

The age of a webpage plays an important part in determining its relevance when displaying search results. The date a webpage has been created or altered can be extracted from its date of last modification, meta-data that is commonly included in HTML webpages. The time elapsed between the day of the webpage's creation/modification and the current day denotes its age. Young webpage, i.e. those that have been created or modified only a little while before being crawled and indexed, are likely to be more up-to-date and informative to a user than old ones. For this purpose, an age filter will extract the age of webpage and assign it a significance ranging from 0 to 1.

The age-significance mapping between these two extremes will be nonlinear and model an inverse relation between age and significance. This means that a more recent webpage will have a higher significance than an older one. It also means that, due to the curved nature of the mapping function, a difference in age between two old webpages will yield a smaller difference in significance than the same difference in age between two younger webpages. The policy can be set to allow only pages with a high significance, or use the significance as an additional heuristic when ranking search results.

4.5 Size

The total size on disk of a webpage can also be used as a factor in determining a page's significance and assign it a significance ranging from 0 to 1. The size-significance mapping between these two extremes will be nonlinear, as is the case in the age filter. Here, however, we expect the function to follow a kind of logarithmic curve, giving larger webpages a higher significance than smaller ones. Smaller webpages may contain little or primarily textual content, while bigger ones may contain rich media such as image, audio, or video. A webpage's size can thus be used as a marker for ranking search results when the search engine is queried from a mobile device.

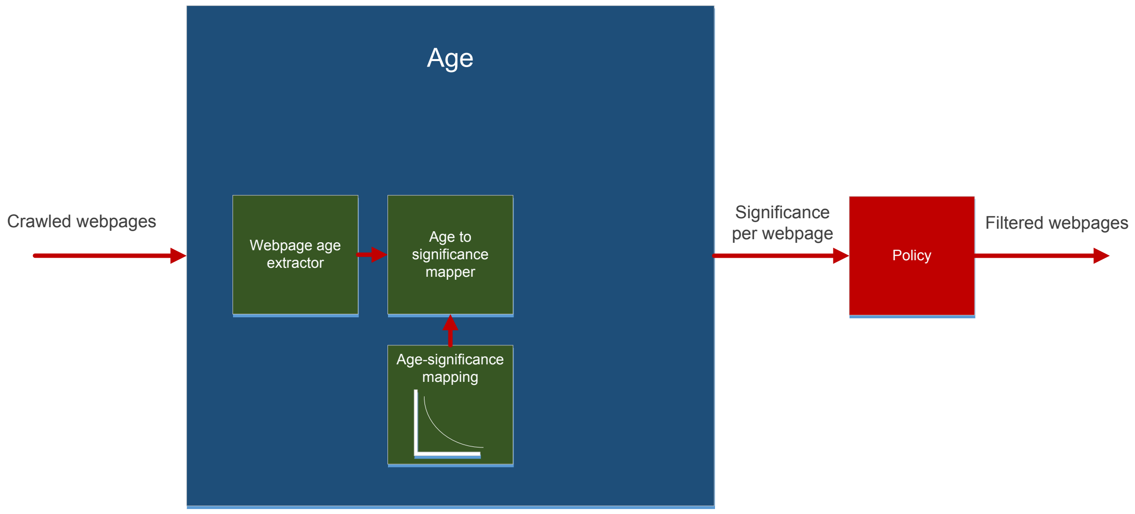


Figure 18: Age

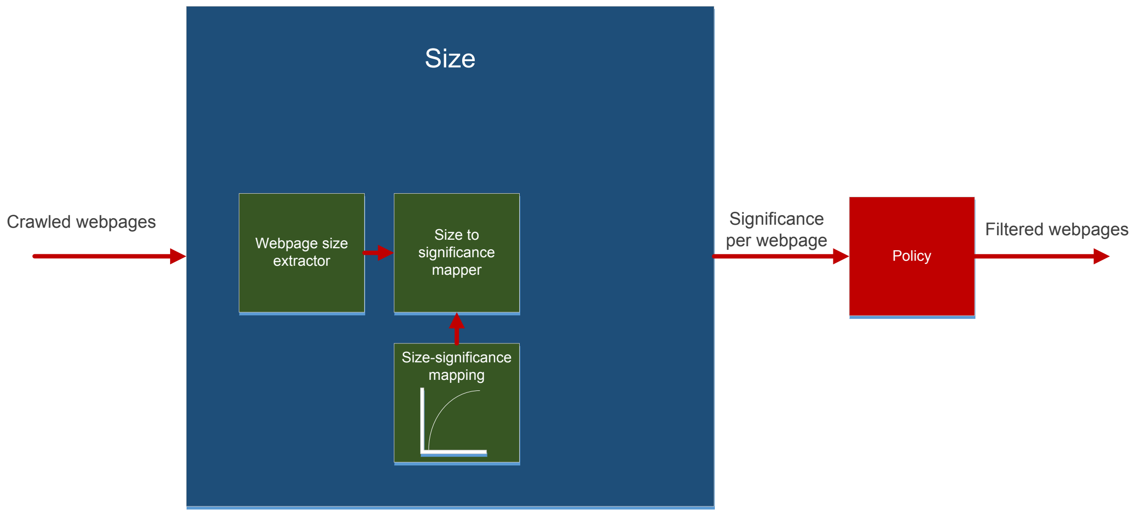


Figure 19: Size

5. Search Management

In order to build an effective search engine a collection of documents and vocabulary are the key elements. Crawlers enable us to build a collection of documents whereas the vocabulary is developed using several techniques that we have seen in Information Management. The collection and vocabulary are then used to build an index, which plays a vital role in searching the right information to fulfill a user need. In this section we shall detail indexing and ranking techniques that are extremely important to evaluate a search query. Figure 20 gives an overview of search management in USE.

The indexer is one of the critical components, which takes its input from multiple sources and it is used to server all kinds of queries. Crawled data is stored in HDFS and indexer uses the file paths in HDFS to be provided later against a keyword. The important decisions that which keywords needs to be indexed comes from the search policy management and keeps on changing over time depending factors like user search preferences. When a query comes in, the query manager processes the query and fetches raw results from the indexer and then ranks them before serving to a specific platform like ordinary browser, cell phone or SMS.

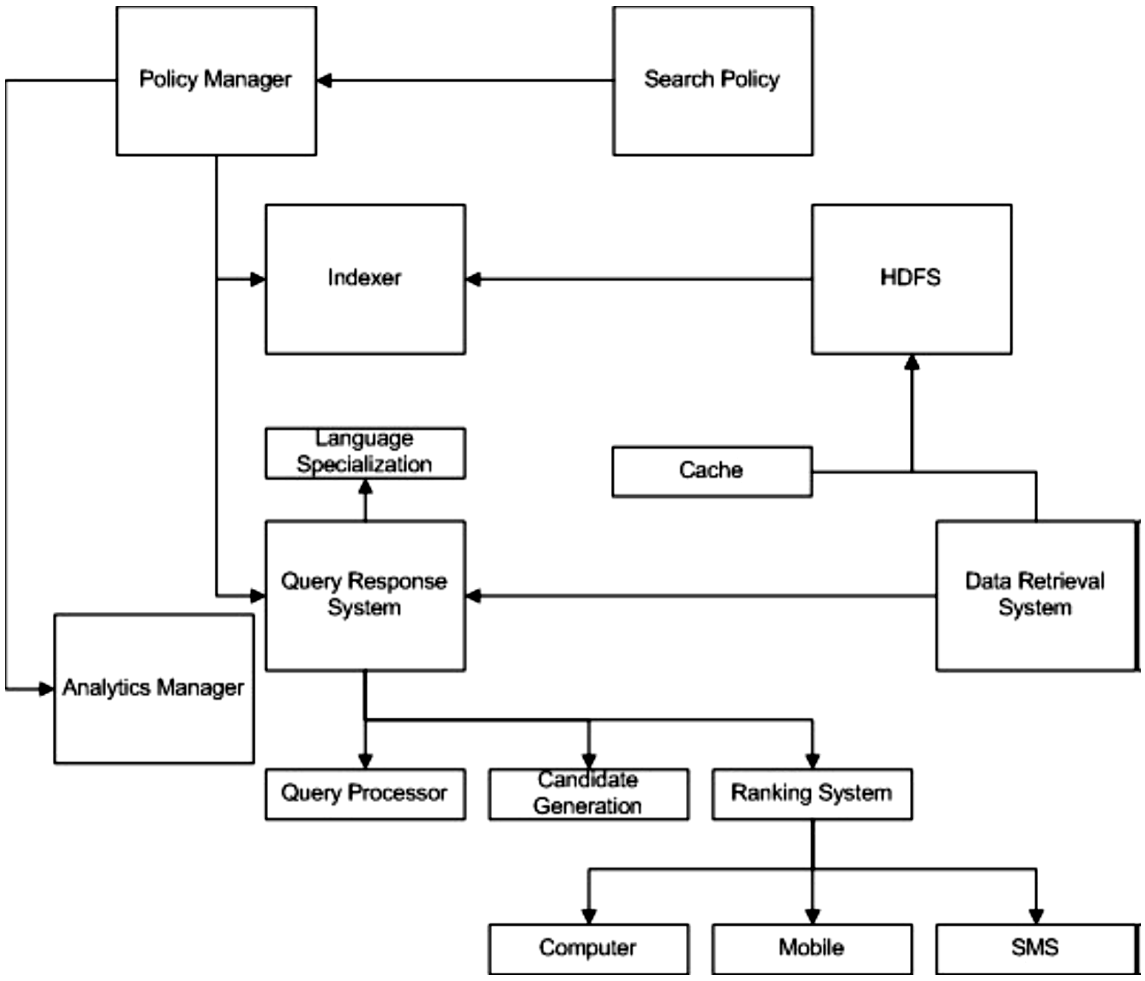


Figure 20: Search Management

5.1 Indexing

The second important component of a search engine after crawler is the Indexer. A crawler fetches the web pages, and the indexer turns them into the inverted index which becomes usable for the search. “An inverted index is able to do many accesses in O(1) time at a price of significantly longer time (O(n)) to do an update. Index construction time is longer as well, but query time is generally faster than with a binary tree. Since index construction is an off-line process, shorter query processing times at the expense of lengthier index construction times is an appropriate tradeoff. Finally, inverted index storage structures can exceed the storage demands of the document collection itself. However, for many systems, the inverted index can be compressed to around ten percent of the original document collection. Search engine developers are happy to trade index construction time and storage for query efficiency. An inverted index is an optimized structure that is built primarily for retrieval, with update being only a secondary consideration. The basic structure inverts the text so that instead of the view obtained from scanning documents where a document is found and then its terms are seen (think of a list of documents each pointing to a list of terms it contains), an index is built that maps terms to documents (pretty much like the index found in the back of this book that maps terms to page numbers). Instead of listing each document once (and each term repeated for each document that contains the term), an inverted index lists each term in the collection only once and then shows a list of all the documents that contain the given term. Each document identifier is repeated for each term that is found in the document.

5.1.1 Solr - Open source Indexing Solutions

‘Solr’ is a Lucene based indexing and search solution, which is getting traction in the market place due to its quality and speed. Solr is a Java servlet based web application. Currently it is a sub-project of Apache’s Lucene project. It has capabilities like full-text search, hit highlighting, search result automatic clustering and user level search filters and handling of rich documents like PDF and MS Word. Additionally it is highly scalable as search query load increases. It achieves this scalability using sharing of search data on multiple machines and index replication.

Like any other open-source project, Solr uses functionality of number of other projects like Carrots2 for clustering and different component of Lucene project (e.g. Tika). The major challenges for our team will be to tailor the indexing system such that, it works effectively with the Urdu language based content. Additionally we will need to tweak clustering and filtering techniques for Urdu language content.

5.2 Implementation Details

Currently we have developed a minimal prototype of Query response system as shown in Figure 21. Frontend of Urdu Search Engine is made to ease Urdu content search. Front end is implemented using HTML, PHP, JavaScript, jQuery and CSS. On user query, frontend code sends a request to Apache Solr and wait for Apache Solr response. On Apache Solr response, search results are formatted and shown to the user.

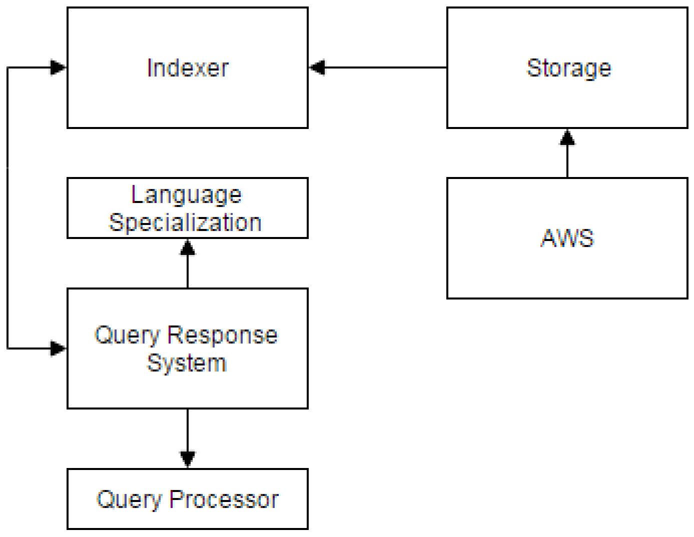


Figure 21: Frontend Implementation

5.2.1 Frontend Infrastructure

We adapted an Ajax-Solr project [19] that loosely follows Model-view-controller (MVC) pattern as shown in figure 22. MVC is a software design pattern for implementing user interfaces on computers. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

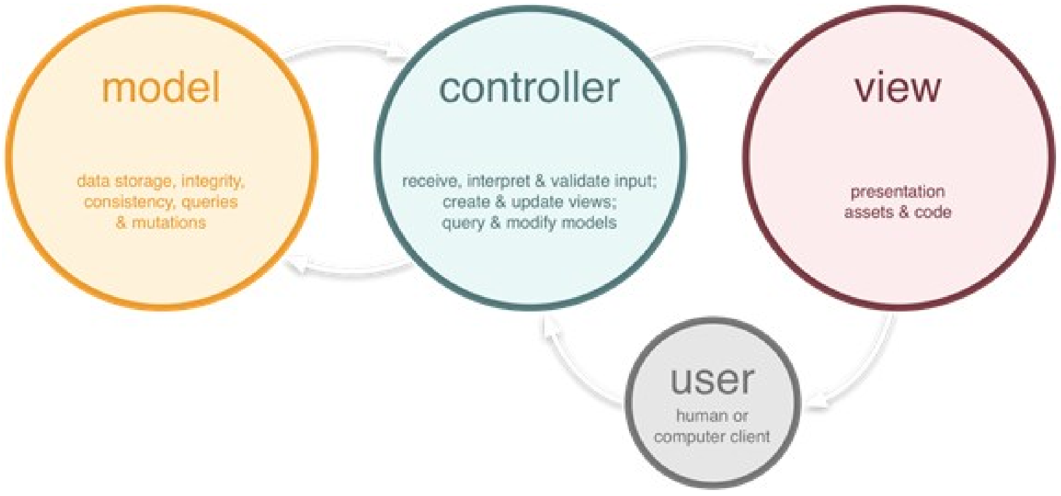

model
data storage, integrity, consistency, queries & mutations
controller
receive, interpret & validate input; create & update views; query & modify models
view
presentation assets & code
user
human or computer client

Figure 22: MVC model

Model: The model directly manages the data, logic, and rules of the application. A model stores data that is retrieved according to commands from the controller and displayed in the view. *ParameterStore* stores the Solr parameters i.e. the state of the application.

View: A view can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. A view generates new output to the user based on changes in the model. Widgets are the views to render different parts of the interface. Solr's response usually consists of hundreds or thousands of documents. Pagination of result set is necessary for search result organization and navigation. Functionality of pagination depends on the number of documents returned by Apache Solr and the URL of different pages of same query results. *js/lib/ajaxsolr/widgets/jquery/Pager.js* file is used for pagination. Various widgets are used to query Solr, prepare response, and display results in the web browser. Widgets are located in *js/lib/nutchsolr/*widgets and important widgets are described in table 4.

Controller: It accepts input and converts it to commands for the model or view. A controller can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., scrolling through a document). Using *ParameterStore*, Manager sends requests to Solr and delegates responses to widgets (for rendering). Manager is the main component used to control all other widgets. In USE frontend code, we are using *js/lib/ajaxsolr/managers/Manager.jquery.js* to prepare final URL submitted to Solr e.g. http://xx.xx.xx.xx:8900/solr/collection1/select?q=english&wt=json&indent=true. Core files are located in *js/lib/ajaxsolr/*core. Important files are described in table 5.

Table 4: Widgets used during implementation

File Name	Description
NResultWidget.js	Used to handle Apache Solr response. Formatting of returned docs depends on group value.
NSearchWidget.js	Used to prepare query before submission to Apache Solr.

Table 5: Core files

File Name	Description
Core.js	Base class for all other classes.
AbstractManager.js	Defines various methods and variables used all over the directory. Manager is the main component for that handles everything.
Parameter.js	Represents a Solr parameter.
ParameterStore.js	Stores Solr parameters and describes their usage.
AbstractWidget.js	Base class for all widgets. Defines parameters for extension.

5.2.2 Internal Working

AJAX Solr is JavaScript framework-agnostic; it only requires an AJAX implementation to send requests to Solr. In this example, we use jQuery. In AJAX Solr, the Manager sends these requests, and passes the responses to each widget for handling. JavaScript files required are Core.js, AbstractManager.js, Manager.jquery.js.

AJAX Solr stores Apache Solr parameters in a ParameterStore. JavaScript files for the ParameterStore are Parameter.js, ParameterStore.js. Next comes the Result Widget to show results. JavaScript files for this purpose are AbstractWidget.js, NResultWidget.js. Next comes the Pager Widget for pagination. JavaScript file for this purpose is Pager.js.

Every widget takes a required id, to identify the widget, and an optional target. The target is usually the CSS selector for the HTML element that the widget updates after each Solr request.

After Request, each widget which runs after the Manager receives the Solr response. The Manager stores the response in “manager.response” (which the widgets may access through this “manager.response”).

The Manager’s *init* method is typically called, to initialize AJAX Solr. This method calls the *init* methods of the ParameterStore and widgets. Nutch.js which is *ajax/js* directory contains Solr URL and various widgets structures.

6. Summary

Research indicates that indigenously developed search engines [20] [21] are more successful in the communities accessing localized content, primarily because they offer language and culture specific services. For example, Google only has 8%, 22% and 31% share in search market in South Korea, China and Japan respectively, till 2012, which is considerably smaller share than the search engines developed locally. As the availability and usage of online content in local language increases, it is becoming essential to provide efficient access to relevant content for Pakistani users through electronic devices including low end mobile phones. This project is developing 'Urdu Search Engine' to address the national and linguistic needs, and to incubate the much needed expertise in this area of research and development. The project is working in three aspects, focusing on high performance distributed computing, content search optimization and local content management.

The report describes the implementation details of the preliminary developments we have done in our 2nd deliverable. As the work has been clearly divided between three teams, each team has done its assigned task in prototype building phase. CI team has prototyped and analyzed the *Common Crawl* data. The analysis showed that the *.pk* domains which makes up 0.01% webpages, is dominated by the *.com* and *.edu* second level domains. Majority of the content on these sites consists of html images and pdf. Our IM team has implemented a prototype for filtering system which is based on linguistic data. As for the front-end of our web interface, our SM team has created a prototype query response system. This prototype system takes query from the user and searches the query in indexed data and return appropriate search results to the user.

Next deliverable will focus on development of crawler and more linguistically challenging tasks like Urdu text summarization. We will also look more closely in the design and implementation of other search engines so we would have better understanding in designing and development process.

References

[1] Cafarella, M. and Cutting, D. Building Nutch: Open Source Search in ACM Queue, 2004, 2.

[2] P. D. Turney. Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In Proceedings of the Twelfth European Conference on Machine Learning, 2001.

[3] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale Information Extraction in KnowItAll. In Proceedings of the 13th International World-Wide Web Conference, 2004.

[4] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates.Unsupervised Named-Entity Extraction from the Web: An Experimental Study. Artificial Intelligence, 2005.

[5] E. Brill, J. Lin, M. Banko, S. T. Dumais, and A. Y. Ng. Data-Intensive Question Answering. In TREC 2001 Proceedings, 2001.

[6] Michael J. Cafarella, Oren Etzioni A Search Engine for Natural Language Applications. Proceedings of the 14th International World Wide Web Conference (WWW 2005).

[7] [Online] Available: http://www.widetutorials.com/seo-search-engine-optimization-for-begineers/

[8] Shvachko, K.; HairongKuang; Radia, S.; Chansler, R.; , "The Hadoop Distributed File System," Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on , vol., no., pp.1-10, 3-7 May 2010.

[9] J. Dean, S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” In Proc. of the 6th Symposium on Operating Systems Design and Implementation, San Francisco CA, Dec. 2004.

[10] Hadoop: Open source implementation of MapReduce. [Online]. Available: http://lucene. apache.org/hadoop/.

[11] Chu, C.-T., Kim, S. K., Lin, Y. A., Yu, Y., Bradski, G., Ng, A., and Olukotun, K. 2006.Map-Reduce for machine learning on multicore. In Proceedings of Neural Information Processing Systems Conference(NIPS). Vancouver, Canada.

[12] "Common Crawl," [Online]. Available: http://commoncrawl.org/.

[13] "Common Crawl October 2016 Corpus," [Online]. Available: http://commoncrawl.org/2016/11/october-2016-crawl-archive-now-available/.

[14] "Common Crawl Files Format," [Online]. Available: http://commoncrawl.org/the-data/get-started/.

[15] "Amazon EMR," [Online]. Available: https://aws.amazon.com/emr/.

[16] "List of datasets," [Online]. Available: http://commoncrawl.org/the-data/get-started/.

[17] "AWS EMR," [Online]. Available: http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-what-is-emr.html.

[18] "Maxmind Homepage," [Online]. Available: https://www.maxmind.com/en/home.

[19] “Ajax-Solr” [Online]. Available: https://github.com/evolvingweb/ajax-solr/wiki/reuters-tutorial

[20] [Online]. Available: http://koreacrunch.com/archive/daum-gains-20-percent-share-in-search-market

[21] [Online]. Available: http://news.ichinastock.com/2011/10/baidu-78-google-18-in-chinas-search-enginemarket-in-q3/