# پاکستان ہمکنار
# Urdu Search Engine

*This document gives detailed description of developments that were made during the **third** milestone of this project. It provides technical progress report on basic crawler development and testing along with code and test case*

High Performance Computing and Networking Lab

Center for Language Engineering

Al-Khawarizmi Institute Of Computer Science,

University Of Engineering and Technology, Lahore

# Basic Crawler Development and Testing Report

# Contents

## Crawler development steps:

A crawler is computer program that repeat some steps again and again to crawl the web. There are some basic steps that are found almost in all crawlers as show in figure 1. These steps are discussed below.

1. List of URLs (seed)
2. Database
3. URLs Selection from database
4. Fetching
5. Parsing
6. Indexing
7. Go to Next Iteration

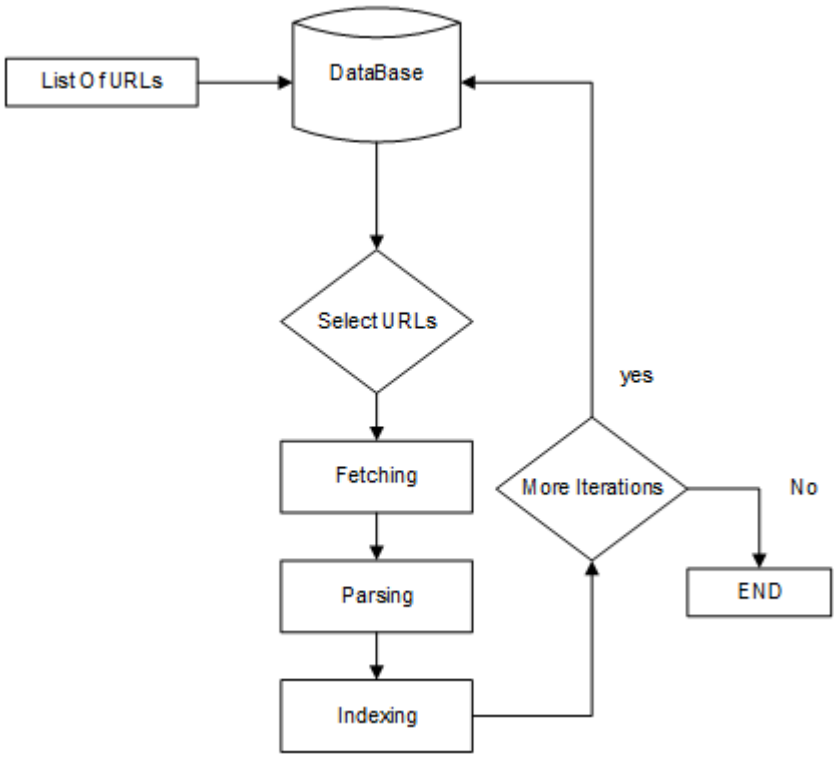


*Figure 1. Basic Crawler Workflow Diagram*

Each of these steps are discussed below.

1. List Of URLs (seed)

A crawler require some URLs at start to crawl the web. These URLs are also called seed. A crawler first fetch these URLs and then according to the configuration, it goes to outer links and inner links of these URLs in next iteration and so on. It is very import to select URLs as seed according to given constraints as crawler is going to expand according to the outer links.
For Our Case, in order to select URLs for Urdu documents, the best URLs will be those having most outer links that are also have Urdu content. But as it is not possible to completely rely on seed URLs where all of outer links are in Urdu, we have to do some post processing to finalize URLs that are going to be fetched. It is very crucial step as if seed is not appropriate ( non-Urdu content) then there is possibility that out links of this URLs will also be in non-Urdu language.

2. Database
   In order to store data and metadata of crawled URLs, a database is required. There a two basic types of database, relational and non-relational. Relational databases like MYSQL, SQLITE etc. represent and store data in rows and columns. Non-relational databases like HBase, Mongo etc. are those databases where there is no relation (joins) between tables. As web data is of different type like html, video, audio, pdfs etc. It is good to use non-relational databases to handle all such type of data. Also as crawler data is going to expand quickly, it is good that storage should be distributed to handle such situation. Crawler have to access the database when it has to select which URLs will be fetched in next iteration. URLs can be out links as well as in links, using some selection or scheduling algorithm, when these URLs will be fetched.
3. URLs selection
   As discussed is last section, this is the main point where we can control which type of web data should be fetched. For a page, there are two type of hyperlinks on it, one is those URLs that have same domain as current page. It is called inner link and second URLs is the one that does not have same name as current page, it is called outer link. In order to crawl few websites only e.g. Wikipedia, only inner links should be allowed by crawler to fetch in each iteration. But it one want to crawl the whole web, crawler should crawl outer links also.
   In order to crawl Urdu web documents only, we have to not only crawl all inner links of given seed but we have to go to outer links also. As it is quite possible that outer links will be non-Urdu pages. Even this situation can happen in inner links. For example bbc.com, this website is basically in English, but there are many other languages option available in home page e.g. bbc.com/URDU, bbc.com/Arabic etc. So have to apply some filter at this stage to sure that URLs are of Urdu language. But before crawling a web page, it is very difficult to say about language of content especially for outer links. So we cannot completely apply filter at this stage. We have to check the content for 100 percent confirmation about the document language. We can apply filters in parsing or indexing stage for this purpose.
4. Fetching
   In this step, URLs are download from web in Batch mode. For this purpose, every language has some API (libraries) available e.g. in python you can use urllib2 or requests libs and in java you can use HttpUrlConnection class. In order to avoid many consecutive requests to same web server, some limit should be applied before next request. If one does not apply a limit to crawler to request to same server, then it is quite possible that there crawler will be blocked soon.
5. Paring

A web page consist of html tags, JavaScript and JQuery code along with original text. In order to extract text from a web page, all these tags should be removed. There are basically two main parts of a web document, one in header that contains title and some metadata tags and second is the body that contains complete page content. By simple intuition, we can assume that title tags contains title of page and content in body tags is the complete text of document. There are many well know parsers available in different languages as well as standalone application e.g. Beautifulsoup (in Python) can parse web document completely and Apache Tika is standalone application used to parse html.

In order to know about different statistics of a web page e.g. language, date etc. Parser can be developed such that along with parsing, it can give you different statistics of page content.

6. Indexing

Indexing is used to quickly search required documents from crawled webpage. It not only optimize speed but performance. Without indexing, a search engine has to scan through all stored documents. There are many different algorithms available that can be used to create an index of crawled webpages. There are some basic steps that are performed on text during indexing. These are briefly discussed here.

6.1. Stop Words Removal

There are some words in each language that are commonly used e.g. in English “the”, “is” etc. Search Engines remove these words at time of search and indexing.

6.2. Tokenization

Before indexing, text of each document is tokenized based on space, comma etc. as delimiter.

6.3. Stemming

Stemming is the process to get the root word of each word e.g. “Retrieval” root word is retrieve. So instead of saving each word, only its stem word is saved. It will not only save the space but also reduce retrieval time. Every language has its own stemmers [1]. In stemming, all words are changed to lower case. It is also called normalization.

6.4. Store the terms

There are many techniques available that are used to store the index. The most common one is called inverted index.

Inverted index is the list of words and the documents (Ids) in which they appear. It can be consider as a tabular form where rows are unique terms (words) and columns names are document unique ids. Data in the cell can be the frequency of occurrence of that word in given documents. There can be the position of word also in each cell.

For better understanding of indexing, let say we have two text documents with following English content.

Document 1: “The quick brown fox jumped over the lazy cat“
Document 2: “Quick brown foxes jump over lazy dogs in summer”

First of all, stop words are removed from these documents. Here “the” and “in” are two stop words. Then they are tokenized and then stem of each word in found. For example dogs stem is dog and foxes stem is fox. Final position of tokens is shown in Table 1. Here table rows are the tokens of text and there are only two columns as there are only two documents. Column names are documents unique ids that can be a URL for web document. When someone search “quick brown fox” then matches will be found.

| Term | Document_1 | Document_2 |
|---|---|---|
| **brown** | X | X |
| **cat** | x | |
| **dog** | | x |
| **fox** | X | X |
| **jump** | X | X |
| **lazy** | X | X |
| **over** | x | x |
| **quick** | x | |
| **summer** | | x |

*Table 1: Inverted Index Example*

For document 1, corresponding score is three as all three search terms exist in it while document 2 score is 2 as only two search terms are found. Simply counting the matching terms we can say that document 1 is more relevant that document 2. There can be different results based on different selection algorithm. For example based on position of one word from other will change the results also [2].

7. Next Iteration
   When indexing is complete, there can be an optional step to remove duplicates if exists. After that crawler goes to step 3 to select new URLs for fetching. This process is continued until a given depth value.

There are many open source crawlers available with different characteristics e.g. scrapy, Apache Nutch, Storm crawler, wget, spider and Opese etc. [3]. In order to crawl whole web, it should be scale able according to the situation. For this scenario, Apache Nutch is the best optional as it can be distributed using Apache Hadoop to as many nodes as required. Also it can store crawled data as distributed key value pair e.g. using Hbase etc. (this support is available is Nutch version 2.x). It has very good architecture to write a new plugin according to new polices. It has also support to index documents to Apache Solr or Elasticsearch [4][5].

## Crawler Testing:

Apache Nutch does not have well documentation available but its simple startup guide can be found on many web sites. There are two Nutch versions available one is 1.x and second is 2.x. Both version are being used for different purposes. The main difference between 1.x and 2.x is that 2.x has the support of Apache Hadoop while 1.x version does not support Hadoop. It stores crawled data in segment on local file system according to given configuration. After research and testing, we have decided to use version 2.x as it can give many other benefits that we cannot obtain in version 1.x for example it supports Hbase that is a distributed key value pair data. For post processing, we can integrate Apache Hive with it. We

can scale up or down Hadoop cluster without interruption of crawling job. Hadoop also provides data reliability i.e. if some nodes (machines) goes down, then it can recover data without any problem by its replication property. To index crawled documents, Apache Solr has been decided to use. It provides full text search. So in order to crawl some web, Hadoop, Hbase and Solr should be configured first. Details of each configuration is given below.

## Apache HADOOP Configuration

In deployment mode, Apache Hadoop is used in distributed mode that requires a lot of machines connected together to make a cluster. It can be used is pseudo distributed mode on single system for testing purpose. Following are the configuration steps used to run Hadoop is pseudo distributed mode.

1. JDK Installation
   In order to run Hadoop, Nutch , Hbase JDK is required. JDK 1.8 was installed in Linux based system with following command.
   *sudo apt-get install openjdk-8-jdk*
2. SSH configuration
   In order to use Hadoop, it is required that user should SSH to localhost (and to other nodes in distributed mode) in passwordless mode. For this purpose, following steps are taken

   ```
   ssh-keygen
   cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
   ssh localhost
   ```

3. Disable IPV6 if enabled
   In order to disable IPV6, /etc/sysctl.conf is updated as following(replaced 0 with 1)

   ```
   net.ipv6.conf.all.disable_ipv6 = 1
   net.ipv6.conf.default.disable_ipv6 = 1
   net.ipv6.conf.lo.disable_ipv6 = 1
   ```

   After that system was rebooted.
4. Download and install Apache Hadoop
   Apache Nutch supports two version of Hadoop as given in documentation 1.2.1 and 2.5.2. In 1.x series, 1.2.1 is the most stable version while 2.x series is most continues for further update in version. We have decided to use version 1.2.1 (we may switch to 2.x series in deployment if required). Its binary format was downloaded from its given repo [6] and extracted to /home/hpcnl/crawler/hadoop-1.2.1 directory.
5. Update .bashrc
   This file is used to set environmental variables that can be used be different application. In order to configure Hadoop, following variables were exported.

   ```
   export HADOOP_HOME=/home/hpcnl/crawler/hadoop-1.2.1
   export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.31-3.b13.fc21.x86_64
   export PATH=$PATH:$HADOOP_HOME/bin
   ```

6. Update Hadoop configuration files

There are 4 Hadoop files that should be updated core-site.xml, hdfs-site.xml, mapred-site.xml and hadoop-env.sh. All these files are placed in $HADOOP_HOME/conf directory.
JAVA_HOME variable was added in hadoop-env.sh as defined in .bashrc file. While remaining 3 files confirmations are updated as given below.

7.1.1. Core-site.xml configuration

```
<property>
<name>hadoop.tmp.dir</name>
<value>/home/hpcnl/hadoop-1.2.1/tmp</value>
<description>A base for other temporary directories.</description>
</property>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:54310</value>
<description>The name of the default file system. A URI whose scheme and authority
determine the FileSystem implementation. The uri's
scheme determines the config property (fs.SCHEME.impl) naming the FileSystem
implementation class. The uri's authority is used to
determine the host, port, etc. for a filesystem.
</description>
</property>
```

And tmp directory is create in $HADOOP home as given in first property.

7.1.2. Mapred-site.xml configuration

```
<property>
<name>mapred.job.tracker</name>
<value>localhost:54311</value>
<description>
The host and port that the MapReduce job tracker runs
at. If "local", then jobs are run in-process as a single map
and reduce task.
</description>
</property>
```

7.1.3. Hdfs-site.xml configuration

```
<property>
<name>dfs.replication</name>
<value>1</value>
<description>Default block replication. The actual number of replications can be specified
when the file is created. The default is used if replication is not specified in create time.
</description>
</property>
```

7. Format NameNode
   In order to start hadoop cluster, first time, its namenode should be formatted.
   $HADOOP_HOME/bin/hadoop namenode -format
8. Start Cluster

In order to start cluster, there are many scripts available in bin directory. One can start few daemons as well as complete cluster. So, in order to run Hadoop all daemons, following command was used.

*$HADOOP_HOME/bin/start-all.sh*

9. Verify all daemons

   There are basically 5 daemons that should be running all the times when cluster is started

   I. NAMENODE
   II. SECONDARYNAMENODE
   III. DATANODE
   IV. JOBTRACKER
   V. TASKTRACKER

   Details of each daemon can be found in appendix A. In order to check services running or not *jps* is used.

## Apache HBASE Configuration

Many version of Hbase are available. Recommended version by Apache Nutch is 0.98.x with hadoop 2.x. For Hadoop version 1.x, recommended version of Hbase was 0.94.x. Similar to Hadoop, Hbase can be configured in pseudo as well as fully distributed mode. Some details about Hbase can be found in Appendix A.

Following steps were taken to configure Hbase in pseudo distributed mode.

1. Download and extraction

   Apache Hbase was download from one of given mirrors and was extracted to /home/hpcnl/hbase-0.94.14

2. Update .bashrc file

   Hbase HOME with bin path was added in .bashrc file.

```
export HBASE_HOME=/home/ghulam/Documents/crawler/hbase-0.94.14
export PATH=$PATH:$HBASE_HOME/bin
```

3. Update some configuration file in conf directory

   I. Update hase-env.sh

```
Export  JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.31-3.b13.fc21.x86_64
export HBASE_MANAGES_ZK=true
```

   II. Update hbase-site.xml

```
<property>
<name>hbase.rootdir</name>
<value>hdfs://localhost:54310/hbase</value>
</property>
<property>
<name>hbase.cluster.distributed</name>
<value>true</value>
</property>
<property>
```

```
<name>hbase.zookeeper.quorum</name>
<value>localhost</value>
</property>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>hbase.zookeeper.property.clientPort</name>
<value>2181</value>
</property>
<property>
<name>hbase.zookeeper.property.dataDir</name>
<value>/home/hpcnl/hbase-0.94.14/zookeeper</value>
</property>
```

4. Start Hbase and verify
   In order to start Hbase, following script was executed.
   `$HBASE_HOME/bin/start-hbase.sh`
   Same command jps was used to verify that all daemons are up. There should three daemons corresponding running on the system, HRegionServer, HMaster and QuorumPeerMain.

## Apache Solr Configuration

Brief introduction of Solr can be found in Appendix A. In our pseudo mode cluster, we are running solr as single node cluster. There are many version of Solr are available till now but Apache Nutch recommends 4.10.3 version of Solr. That's why we have decided to use this version in our testing setup. Follow steps are taken to configure and start solr service.

1. Download and extract
   From one of given Solr mirrors, 4.10.3 versions was download and extracted.
2. Schema file is $SOLR_HOME/example/solr/collection1/conf/ is replaced with the schema file provided by Apache Nutch. Schema.xml file is also given in Appendix A.
3. Run Start.jar (to detach session, use run command in screen)
   Java –jar start.jar

## Apache Nutch Configuration and Testing with Hadoop cluster

Some introduction and phases details of Apache Nutch can be found in Appendix A. Following configuration of Apache Nutch was updated to run a test case.

i. Download and install
   Apache Nutch 2.x is available in source only. One has to compile it after downloading. After downloading, following steps are taken to compile the code.
   1. Download a source package
   2. extract it using tar command and cd in its directory
   3. Edit conf/nutch-site.xml

```
<property>
  <name>http.agent.name</name>
```

```
  <value>USE-crawler</value>
</property>
<property>
  <name>storage.data.store.class</name>
  <value>org.apache.gora.hbase.store.HBaseStore</value>
  <description>Default class for storing data</description>
</property>
<property>
 <name>plugin.includes</name>
 <value>protocol-httpclient|protocol-http|indexer-solr|urlfilter-regex|parse-
(html|tika)|index-(basic|more)|urlnormalizer-(pass|regex|basic)|scoring-opic</value>
</property>
<property>
<name>parser.character.encoding.default</name>
<value>utf-8</value>
<description>The character encoding to fall back to when no other information is
available</description>
</property>
<property>
 <name>http.robots.403.allow</name>
 <value>true</value>
</property>
<property>
 <name>db.max.outlinks.per.page</name>
 <value>-1</value>
</property>
<property>
 <name>http.robots.agents</name>
 <value>USE-crawler,*</value>
</property>
```

4. Edit ivy/ivy.xml , Scroll down to section Gora artifacs and uncomment this line:

```
<dependency org="org.apache.gora" name="gora-hbase" rev="0.3" conf="*->default" />
```

(It is assumed that we are using hbase)

5. Now Compile using ANT. ( For compilation Internet is required)
   Go to $NUTCH_HOME and run below command
   Ant runtime
6. After successful compilation, runtime directory is created that contains two subdirectory local and deploy. In order to run Nutch on single system (pseudo distributed), local directroy setup has been used.

ii. Running first job

In order to run Nutch, first all Hadoop, Hbase daemons should be running along with Solr to index crawled data. First of a seed is required. It was decided to crawl dawnnews.tv that is an Urdu website. Following steps are carried out to run our job.

1. Add test URL to seed file
   cd apache-nutch-2.3.1/runtime/local/
   mkdir urls
   echo "http://dawnnews.tv" > urls/seed.txt

2. Inject phase. (it will add given URL to Hbase table)
   bin/nutch inject urls/seed.txt -crawlId pk
3. Generate Phase. ( It will mark URLs for fetching i.e. URLs selection phase)
   bin/nutch generate -topN 9 -crawlId pk
4. Fetching Phase
   bin/nutch fetch -all
5. Parsing Phase
   bin/nutch parse –all
6. UpdateDb (Hbase table update)
   bin/nutch updatedb
7. Indexing to Solr
   bin/nutch solrindex http://localhost:8983/solr/ -all -crawlId pk

iii. Check Indexed documents to Solr

In order to test Solr for new document crawled, following query was sent.
http://localhost:8983/solr/collection1/select?q=سیاسی+مصر و فیات&wt=json&indent=true&fl=content&df=content
Which Returns following result

<?xml version="1.0" encoding="UTF-8" ?>

- <response>

+ <lst name="**responseHeader**">

- <result name="**response**" numFound="**1**" start="**0**">

- <doc>

<str name="**title**">**Home - Dawn News**</str>

<str name="**url**">**https://www.dawnnews.tv/** </str>

<str name="content">Home – Dawn News LIVE TV Dawn.COM Images Herald Aurora CityFM89 Events Supplements Classifieds Obituaries Dawn News Television Advertisement پہلا صفحہ پاکستان کا کابل اور دہلی کے ساتھ واٹر مینجمنٹ پر غور یہ پالیسی مشترکہ مفادات کونسل کے اجلاس کے ایجنڈے میں شامل تھی

دہشتگردوں کو پھانسی دہشت گرد سیکیورٹی اہلکار، شہریوں پر حملے اور 4 فوجی عدالتوں سے سزا پانے والے 03,201711:25am تاہم وزیر اعظم کی سیاسی مصروفیات کے سبب اسے مؤخر کر دیا گیا۔ اپ ڈیٹ ئ
افراد ہلاک عینی شاہدین کے مطابق دھماکا خود کش تھا، جس کے نتیجے میں قریب کھڑی 8 کابل میں نیٹو قافلے کے قریب دھماکا، 03,201710:17am خطرناک جرائم میں ملوث تھے، آئی ایس پی آر اپ ڈیٹ ئ
امریکا کا پاک-بھارت براہ راست مذاکرات پر زور ہمارا ماننا ہے کہ عملی تعاون ہی پاکستان اور بھارت کے مفاد میں ہے، 03,201710:24am عام شہریوں کی گاڑیوں کو بھی شدید نقصان پہنچا۔ اپ ڈیٹ ئ
نواز شریف کا استعفیٰ مانگنے والوں کو دھول چاٹنا پڑے گی' پاناما پیپرز کا تعلق کرپشن سے نہیں تھا جسے چور اور لٹیرے بھی تسلیم کرتے ہیں،' 03,201711:16am ترجمان امریکی اسٹیٹ ڈپارٹمنٹ اپ ڈیٹ ئ
ایبٹ آباد کمیشن کی رپورٹ منظوری کے بعد جاری ہو گی' کمیشن کی رپورٹ میں قومی سلامتی سے متعلق چند حساس معلومات بھی شامل ہیں، مریم' 03,201711:01am مریم نواز کا ٹوئٹر پیغام اپ ڈیٹ ئ
عمران خان کے رویے کے خلاف سندھ اسمبلی میں قرارداد منظور پاکستان مسلم لیگ (ن) 03,201709:58am اورنگ زیب اپ ڈیٹ ئ </str>

</doc></result></response>

As currently there was only single page in solr, that's why we got one result according to the query.

## Appendix A:

## Apache Hadoop

Hadoop is an open source, Java-based programming framework that supports the processing and storage of extremely large data sets in a distributed computing environment. It is part of the Apache project sponsored by the Apache Software Foundation [11].

Hadoop makes it possible to run applications on systems with thousands of commodity hardware nodes, and to handle thousands of terabytes of data. Its distributed file system facilitates rapid data transfer rates among nodes and allows the system to continue operating in case of a node failure. This approach lowers the risk of catastrophic system failure and unexpected data loss, even if a significant number of nodes become inoperative. Consequently, Hadoop quickly emerged as a foundation for big data processing tasks, such as scientific analytics, business and sales planning, and processing enormous volumes of sensor data, including from internet of things sensors [12].

**Hadoop Distributed File System (HDFS):**

The primary objective of HDFS is to store data reliably even in the presence of failures including NameNode failures, DataNode failures and network partitions. The NameNode is a single point of failure for the HDFS cluster and a DataNode stores data in the Hadoop file management system.

HDFS uses a master/slave architecture in which one device (the master) controls one or more other devices (the slaves). The HDFS cluster consists of a single NameNode and a master server manages the file system namespace and regulates access to file [14].

**NameNode:**

NameNode is the centerpiece of HDFS. It is also known as the Master. NameNode only stores the metadata of HDFS – the directory tree of all files in the file system, and tracks the files across the cluster. It does not store the actual data or the dataset. The data itself is actually stored in the DataNodes. It knows the list of the blocks and its location for any given file in HDFS. With this information NameNode knows how to construct the file from blocks. NameNode is so critical to HDFS and when the NameNode is down, HDFS/Hadoop cluster is inaccessible and considered down. NameNode is a single point of failure in Hadoop cluster. It is usually configured with a lot of memory (RAM). Because the block locations are help in main memory [13].

**DataNode**

DataNode is responsible for storing the actual data in HDFS. It is also known as the Slave. NameNode and DataNode are in constant communication. When a DataNode starts up it announce itself to the NameNode along with the list of blocks it is responsible for but when a DataNode is down, it does not affect the availability of data or the cluster. NameNode will arrange for replication for the blocks managed by the DataNode that is not available. DataNode is usually configured with a lot of hard disk space. Because the actual data is stored in the DataNode [13].

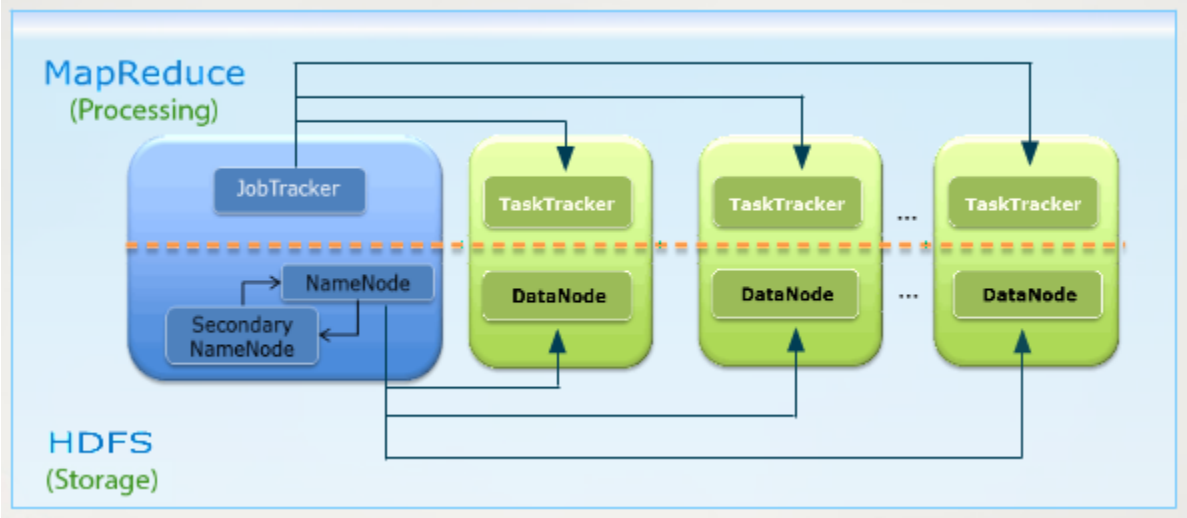


*Figure 1. Hadoop Daemons*

**Secondary NameNode:**
The NameNode stores modifications to the file system as a log appended to a native file system file, edits. When a NameNode starts up, it reads HDFS state from an image file, fsimage, and then applies edits from the edits log file. It then writes new HDFS state to the fsimage and starts normal operation with an empty edits file. Since NameNode merges fsimage and edits files only during start up, the edits log file could get very large over time on a busy cluster. Another side effect of a larger edits file is that next restart of NameNode takes longer.

The secondary NameNode merges the fsimage and the edits log files periodically and keeps edits log size within a limit. It is usually run on a different machine than the primary NameNode since its memory requirements are on the same order as the primary NameNode [12].

**Hadoop Map Reduce:**
The primary objective of Map/Reduce is to split the input data set into independent chunks that are processed in a completely parallel manner. The Hadoop MapReduce framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically, both the input and the output of the job are stored in a file system.

**JobTracker:**

Job Tracker keeps track of all the MapReduces jobs that are running on various nodes. This schedules the jobs, keeps track of all the map and reduce jobs running across the nodes. If any one of those jobs fails, it reallocates the job to another node, etc. In simple terms, JobTracker is responsible for making sure that the query on a huge dataset runs successfully and the data is returned to the client in a reliable manner.

**TaskTracker:**

TaskTracker performs the map and reduce tasks that are assigned by the JobTracker. TaskTracker also constantly sends a hearbeat message to JobTracker, which helps JobTracker to decide whether to delegate a new task to this particular node or not.

## Apache Hbase

HBase is a column-oriented database management system that runs on top of HDFS. It is well suited for sparse data sets, which are common in many big data use cases. Unlike relational database systems, HBase does not support a structured query language like SQL; in fact, HBase isn't a relational data store at all. HBase applications are written in Java much like a typical MapReduce application.

An HBase system comprises a set of tables. Each table contains rows and columns, much like a traditional database. Each table must have an element defined as a Primary Key, and all access attempts to HBase tables must use this Primary Key. An HBase column represents an attribute of an object; for example, if the table is storing diagnostic logs from servers in your environment, where each row might be a log record, a typical column in such a table would be the timestamp of when the log record was written, or perhaps the server name where the record originated. In fact, HBase allows for many attributes to be grouped together into what are known as column families, such that the elements of a column family are all stored together. This is different from a row-oriented relational database, where all the columns of a given row are stored together.

With HBase you must predefine the table schema and specify the column families. However, it's very flexible in that new columns can be added to families at any time, making the schema flexible and therefore able to adapt to changing application requirements.

Just as HDFS has a NameNode and slave nodes, and MapReduce has JobTracker and TaskTracker slaves, HBase is built on similar concepts. In HBase a master node manages the cluster and region servers store portions of the tables and perform the work on the data.

All the webpages and Meta data which Apache Nutch crawler crawl is stored in the hbase tables. This data is then used for further analysis, categorizing, extracting Meta information and sometimes for updating the crawled data [7][8].

## Apache Nutch

Apache Nutch is a highly extensible and scalable open source web crawler software project. Stemming from Apache Lucene, the project has diversified and now comprises two codebases, namely:

- Nutch 1.x: A well matured, production ready crawler. 1.x enables fine grained configuration, relying on Apache Hadoop data structures, which are great for batch processing.

- Nutch 2.x: An emerging alternative taking direct inspiration from 1.x, but which differs in one key area; storage is abstracted away from any specific underlying data store by using Apache Gora for handling object to persistent mappings. This means we can implement an extremely flexibile model/stack for storing everything (fetch time, status, content, parsed text, outlinks, inlinks, etc.) into a number of NoSQL storage solutions.

## Nutch Phases of crawling:

Below are the phases of crawling [10]:

**Create Seed list:**

A URL seed list includes a list of websites, one-per-line, which Apache Nutch will look to crawl.

**Injectector job:**
Injector job is to get URLs for crawling from the file seed.txt and apply filters (if any in *conf/regex-urlfilter.txt*).

**Generator job:**
Generator job is to select URLs for fetch job. Generating batchID (timestamp. any random no.) containing URLs.

**Fetcher Job:**
Fetcher job is to fetch the URLs. Means to get html page of specific url.

**Parser Job:**
HTML parser and Apache Tika parser is used for parsing the fetched content like getting title from title tags of html page.

**DbUpdater Job:**
DbUpdater update the crawl DB with new URLs (out links) fetched from crawling.

**IndexingJob:**
Index the crawled data to Solr (or elasticsearch). It copies the fields from Nutch to Solr.

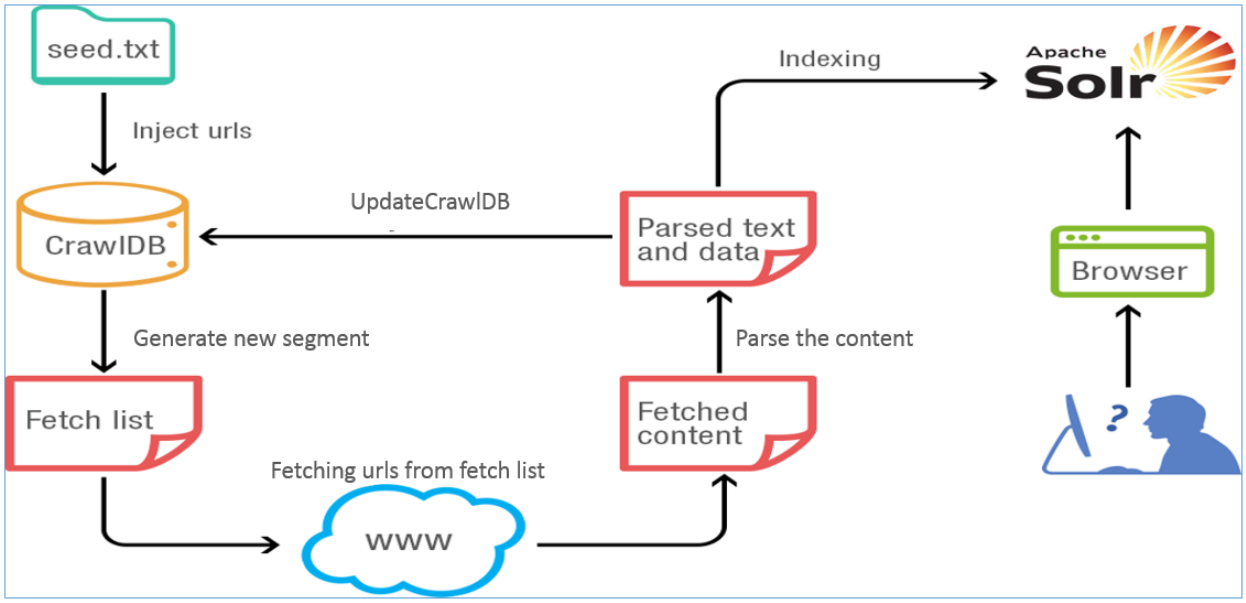


*Figure 2. Apache Nutch Crawling Phases*

## Apache Solr

Apache Solr is an open source enterprise search platform, written in Java, from the Apache Lucene project. Its major features include full-text search, hit highlighting, faceted search, real-time indexing, dynamic clustering, database integration, NoSQL features and rich document (e.g., Word, PDF) handling. Providing distributed search and index replication, Solr is designed for scalability and fault tolerance. Solr is the second-most popular enterprise search engine after Elasticsearch.

Solr runs as a standalone full-text search server. It uses the Lucene Java search library at its core for full-text indexing and search, and has REST-like HTTP/XML and JSON APIs that make it usable from most popular programming languages. Solr's external configuration allows it to be tailored to many types of application without Java coding, and it has a plugin architecture to support more advanced customization.

Solr is bundled as the built-in search in many applications such as CMS/ECM systems. The major Hadoop distributions from Cloudera, Hortonworks and MapR all bundle Solr as the search engine for their Big Data platforms. Solr is supported as an end point in various data processing frameworks and Enterprise integration frameworks.

### Schema.xml

This is the main configuration file is solr. Here you can define fields just like in relational databases. For example, possible fields for a web page can be title, content, text, URL, date, language, content-type etc. Apache Nutch a schema file that is better option to use in solr schema instead of defining each field again. You have to copy schema.xml file from Nutch conf directory to Solr conf directory ( in corresponding core).

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
- <schema name="nutch" version="1.5">
- <types>
  - <!-- The StrField type is not analyzed, but indexed/stored verbatim. -->
  <fieldType name="string" class="solr.StrField" sortMissingLast="true" omitNorms="true" />
  - <!-- <similarity class="org.apache.lucene.search.similarities.DefaultSimilarity"/> -->
    - <!-- Default numeric field types. For faster range queries, consider the tint/tfloat/tlong/tdouble types. -->
  <fieldType name="int" class="solr.TrieIntField" precisionStep="0" omitNorms="true" positionIncrementGap="0" />
  <fieldType name="float" class="solr.TrieFloatField" precisionStep="0" omitNorms="true" positionIncrementGap="0" />
  <fieldType name="long" class="solr.TrieLongField" precisionStep="0" omitNorms="true" positionIncrementGap="0" />
  <fieldType name="double" class="solr.TrieDoubleField" precisionStep="0" omitNorms="true" positionIncrementGap="0" />
    - <!-- Numeric field types that index each value at various levels of precision
         to accelerate range queries when the number of values between the range
         endpoints is large. See the javadoc for NumericRangeQuery for internal implementation details.

         Smaller precisionStep values (specified in bits) will lead to more tokens
         indexed per value, slightly larger index size, and faster range queries.
         A precisionStep of 0 disables indexing at different precision levels.
    -->
  <fieldType name="tint" class="solr.TrieIntField" precisionStep="8" omitNorms="true" positionIncrementGap="0" />
  <fieldType name="tfloat" class="solr.TrieFloatField" precisionStep="8" omitNorms="true" positionIncrementGap="0" />
  <fieldType name="tlong" class="solr.TrieLongField" precisionStep="8" omitNorms="true" positionIncrementGap="0" />
  <fieldType name="tdouble" class="solr.TrieDoubleField" precisionStep="8" omitNorms="true" positionIncrementGap="0" />
    - <!--
       The format for this date field is of the form 1995-12-31T23:59:59Z, and
```

```
                    is a more restricted form of the canonical representation
    of dateTime
             http://www.w3.org/TR/xmlschema-2/#dateTime
             The trailing "Z" designates UTC time and is mandatory.
             Optional fractional seconds are allowed: 1995-12-31T23:59:59.999Z
             All other components are mandatory.

             Expressions can also be used to denote calculations that should be
             performed relative to "NOW" to determine the value, ie...

                   NOW/HOUR
                      ... Round to the start of the current hour
                   NOW-1DAY
                      ... Exactly 1 day prior to now
                   NOW/DAY+6MONTHS+3DAYS
                      ... 6 months and 3 days in the future from the start of
                          the current day

             Consult the DateField javadocs for more information.

             Note: For faster range queries, consider the tdate type

   -->
<fieldType name="date" class="solr.TrieDateField" omitNorms="true" precisionStep="0"
    positionIncrementGap="0" />
- <!--
  A Trie based date field for faster date range queries and date faceting.
   -->
<fieldType name="tdate" class="solr.TrieDateField" omitNorms="true" precisionStep="6"
    positionIncrementGap="0" />
     - <!--
      solr.TextField allows the specification of custom text analyzers
                    specified as a tokenizer and a list of token filters.
    Different
             analyzers may be specified for indexing and querying.

             The optional positionIncrementGap puts space between multiple
    fields of
             this type on the same document, with the purpose of preventing
    false phrase
             matching across fields.

             For more info on customizing your analyzer chain, please see
             http://wiki.apache.org/solr/AnalyzersTokenizersTokenFilters

   -->
     - <!--
      A general text field that has reasonable, generic
```

```
                    cross-language defaults: it tokenizes with
      StandardTokenizer,
            removes stop words from case-insensitive "stopwords.txt"
            (empty by default), and down cases.  At query time only, it
            also applies synonyms.
    -->
- <fieldType name="text_general" class="solr.TextField" positionIncrementGap="100">
- <analyzer type="index">
- <similarity class="solr.DFRSimilarityFactory">
  <str name="basicModel">I(F)</str>
  <str name="afterEffect">B</str>
  <str name="normalization">H2</str>
    </similarity>
  <tokenizer class="solr.StandardTokenizerFactory" />
  <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"
    enablePositionIncrements="true" />
      - <!--
       in this example, we will only use synonyms at query time
                          <filter class="solr.SynonymFilterFactory"
      synonyms="index_synonyms.txt" ignoreCase="true" expand="false"/>
    -->
  <filter class="solr.LowerCaseFilterFactory" />
    </analyzer>
- <analyzer type="query">
  <tokenizer class="solr.StandardTokenizerFactory" />
  <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"
    enablePositionIncrements="true" />
  <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true"
    expand="true" />
  <filter class="solr.LowerCaseFilterFactory" />
    </analyzer>
    </fieldType>
      - <!--
       A text field with defaults appropriate for English: it
```

```
                    tokenizes with StandardTokenizer, removes English stop
    words
            (stopwords.txt), down cases, protects words from protwords.txt,
    and
            finally applies Porter's stemming.  The query time analyzer
            also applies synonyms from synonyms.txt.
  -->
- <fieldType name="text_en" class="solr.TextField" positionIncrementGap="100">
- <analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory" />
    - <!--
     in this example, we will only use synonyms at query time
                      <filter class="solr.SynonymFilterFactory"
    synonyms="index_synonyms.txt" ignoreCase="true" expand="false"/>
  -->
    - <!--
     Case insensitive stop word removal.
                        add enablePositionIncrements=true in both the index
    and query
             analyzers to leave a 'gap' for more accurate phrase queries.
  -->
  <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"
    enablePositionIncrements="true" />
  <filter class="solr.LowerCaseFilterFactory" />
  <filter class="solr.EnglishPossessiveFilterFactory" />
  <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt" />
    - <!--
     Optionally you may want to use this less aggressive stemmer instead of
    PorterStemFilterFactory:
                    <filter class="solr.EnglishMinimalStemFilterFactory"/>
  -->
  <filter class="solr.PorterStemFilterFactory" />
  </analyzer>
- <analyzer type="query">
  <tokenizer class="solr.StandardTokenizerFactory" />
```

```
<filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true"
  expand="true" />
<filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"
  enablePositionIncrements="true" />
<filter class="solr.LowerCaseFilterFactory" />
<filter class="solr.EnglishPossessiveFilterFactory" />
<filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt" />
    - <!--
     Optionally you may want to use this less aggressive stemmer instead of
    PorterStemFilterFactory:
                      <filter class="solr.EnglishMinimalStemFilterFactory"/>
  -->
<filter class="solr.PorterStemFilterFactory" />
  </analyzer>
  </fieldType>
    - <!--
     A text field with defaults appropriate for English, plus
                  aggressive word-splitting and autophrase features enabled.
          This field is just like text_en, except it adds
          WordDelimiterFilter to enable splitting and matching of
          words on case-change, alpha numeric boundaries, and
          non-alphanumeric chars.  This means certain compound word
          cases will work, for example query "wi fi" will match
          document "WiFi" or "wi-fi".  However, other cases will still
          not match, for example if the query is "wifi" and the
          document is "wi fi" or if the query is "wi-fi" and the
          document is "wifi".
  -->
- <fieldType name="text_en_splitting" class="solr.TextField" positionIncrementGap="100"
  autoGeneratePhraseQueries="true">
- <analyzer type="index">
<tokenizer class="solr.WhitespaceTokenizerFactory" />
    - <!--
     in this example, we will only use synonyms at query time
                        <filter class="solr.SynonymFilterFactory"
    synonyms="index_synonyms.txt" ignoreCase="true" expand="false"/>
  -->
```

```
- <!--
    Case insensitive stop word removal.
                        add enablePositionIncrements=true in both the index
  and query
            analyzers to leave a 'gap' for more accurate phrase queries.
  -->
  <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"
    enablePositionIncrements="true" />
  <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
    generateNumberParts="1" catenateWords="1" catenateNumbers="1" catenateAll="0"
    splitOnCaseChange="1" />
  <filter class="solr.LowerCaseFilterFactory" />
  <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt" />
  <filter class="solr.PorterStemFilterFactory" />
    </analyzer>
- <analyzer type="query">
  <tokenizer class="solr.WhitespaceTokenizerFactory" />
  <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true"
    expand="true" />
  <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"
    enablePositionIncrements="true" />
  <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
    generateNumberParts="1" catenateWords="0" catenateNumbers="0" catenateAll="0"
    splitOnCaseChange="1" />
  <filter class="solr.LowerCaseFilterFactory" />
  <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt" />
  <filter class="solr.PorterStemFilterFactory" />
    </analyzer>
    </fieldType>
    - <!--
      Less flexible matching, but less false matches.  Probably not ideal for
    product names,
                      but may be good for SKUs.  Can insert dashes in the wrong
    place and still match.
  -->
```

```
- <fieldType name="text_en_splitting_tight" class="solr.TextField"
    positionIncrementGap="100" autoGeneratePhraseQueries="true">
- <analyzer>
  <tokenizer class="solr.WhitespaceTokenizerFactory" />
  <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true"
    expand="false" />
  <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
  <filter class="solr.WordDelimiterFilterFactory" generateWordParts="0"
    generateNumberParts="0" catenateWords="1" catenateNumbers="1" catenateAll="0" />
  <filter class="solr.LowerCaseFilterFactory" />
  <filter class="solr.KeywordMarkerFilterFactory" protected="protwords.txt" />
  <filter class="solr.EnglishMinimalStemFilterFactory" />
      - <!--
       this filter can remove any duplicate tokens that appear at the same
      position - sometimes
                                possible with WordDelimiterFilter in conjuncton
      with stemming.
    -->
  <filter class="solr.RemoveDuplicatesTokenFilterFactory" />
    </analyzer>
    </fieldType>
      - <!--
       Just like text_general except it reverses the characters of
                    each token, to enable more efficient leading wildcard
      queries.
    -->
- <fieldType name="text_general_rev" class="solr.TextField"
    positionIncrementGap="100">
- <analyzer type="index">
  <tokenizer class="solr.StandardTokenizerFactory" />
  <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"
    enablePositionIncrements="true" />
  <filter class="solr.LowerCaseFilterFactory" />
  <filter class="solr.ReversedWildcardFilterFactory" withOriginal="true"
    maxPosAsterisk="3" maxPosQuestion="2" maxFractionAsterisk="0.33" />
```

```
    </analyzer>
- <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory" />
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true"
      expand="true" />
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"
      enablePositionIncrements="true" />
    <filter class="solr.LowerCaseFilterFactory" />
    </analyzer>
    </fieldType>
- <fieldtype name="phonetic" stored="false" indexed="true" class="solr.TextField">
- <analyzer>
    <tokenizer class="solr.StandardTokenizerFactory" />
    <filter class="solr.DoubleMetaphoneFilterFactory" inject="false" />
    </analyzer>
    </fieldtype>
- <fieldtype name="payloads" stored="false" indexed="true" class="solr.TextField">
- <analyzer>
    <tokenizer class="solr.WhitespaceTokenizerFactory" />
      - <!--
                        The DelimitedPayloadTokenFilter can put payloads on
    tokens... for example,
            a token of "foo|1.4"  would be indexed as "foo" with a payload of
    1.4f
            Attributes of the DelimitedPayloadTokenFilterFactory :
             "delimiter" - a one character delimiter. Default is | (pipe)
           "encoder" - how to encode the following value into a playload
              float -> org.apache.lucene.analysis.payloads.FloatEncoder,
              integer -> o.a.l.a.p.IntegerEncoder
              identity -> o.a.l.a.p.IdentityEncoder
                Fully Qualified class name implementing PayloadEncoder, Encoder
    must have a no arg constructor.
    -->
    <filter class="solr.DelimitedPayloadTokenFilterFactory" encoder="float" />
    </analyzer>
```

```
    </fieldtype>
  - <!--
    lowercases the entire field value, keeping it as a single token.
    -->
- <fieldType name="lowercase" class="solr.TextField" positionIncrementGap="100">
- <analyzer>
  <tokenizer class="solr.KeywordTokenizerFactory" />
  <filter class="solr.LowerCaseFilterFactory" />
    </analyzer>
    </fieldType>
- <fieldType name="url" class="solr.TextField" positionIncrementGap="100">
- <analyzer>
  <tokenizer class="solr.StandardTokenizerFactory" />
  <filter class="solr.LowerCaseFilterFactory" />
  <filter class="solr.WordDelimiterFilterFactory" generateWordParts="1"
    generateNumberParts="1" />
    </analyzer>
    </fieldType>
- <fieldType name="text_path" class="solr.TextField" positionIncrementGap="100">
- <analyzer>
  <tokenizer class="solr.PathHierarchyTokenizerFactory" />
    </analyzer>
    </fieldType>
  - <!--
    My custom field for dictionary implementation
    -->
- <fieldType name="exactstring" class="solr.TextField" sortMissingLast="true"
    omitNorms="true">
- <analyzer type="query">
  <tokenizer class="solr.KeywordTokenizerFactory" />
    </analyzer>
```

```
    </fieldType>
  - <!--
     End of dictionary implementation
     -->
       - <!--
          since fields of this type are by default not stored or indexed,
                        any data added to them will be ignored outright.
     -->
    <fieldtype name="ignored" stored="false" indexed="false" multiValued="true"
      class="solr.StrField" />
     </types>
  - <!--
     <similarity class="solr.SchemaSimilarityFactory"/>
     -->
  - <fields>
    <field name="id" type="string" stored="true" indexed="true" required="true" />
  - <!--
     core fields
     -->
    <field name="batchId" type="string" stored="true" indexed="false" />
    <field name="digest" type="string" stored="true" indexed="false" />
    <field name="boost" type="float" stored="true" indexed="false" />
  - <!--
     fields for index-basic plugin
     -->
    <field name="host" type="url" stored="false" indexed="true" />
    <field name="url" type="url" stored="true" indexed="true" />
    <field name="orig" type="url" stored="true" indexed="true" />
  - <!--
     stored=true for highlighting, use term vectors  and positions for fast
      highlighting
     -->
    <field name="content" type="text_general" stored="true" indexed="true" />
    <field name="title" type="text_general" stored="true" indexed="true" />
```

```
<field name="cache" type="string" stored="true" indexed="false" />

<field name="tstamp" type="date" stored="true" indexed="false" default="NOW" />

- <!--

 catch-all field
  -->

<field name="text" type="text_general" stored="false" indexed="true" multiValued="true"
  />

- <!--

 fields for index-anchor plugin
  -->

<field name="anchor" type="text_general" stored="true" indexed="true"
  multiValued="true" />

- <!--

 fields for index-more plugin
  -->

<field name="type" type="string" stored="true" indexed="true" multiValued="true" />

<field name="contentLength" type="string" stored="true" indexed="false" />

<field name="lastModified" type="date" stored="true" indexed="false" />

<field name="date" type="tdate" stored="true" indexed="true" />

- <!--

 fields for languageidentifier plugin
  -->

<field name="lang" type="string" stored="true" indexed="true" />

- <!--

 fields for subcollection plugin
  -->

<field name="subcollection" type="string" stored="true" indexed="true"
  multiValued="true" />

- <!--

 fields for feed plugin (tag is also used by microformats-reltag)
  -->

<field name="author" type="string" stored="true" indexed="true" />

<field name="tag" type="string" stored="true" indexed="true" multiValued="true" />

<field name="feed" type="string" stored="true" indexed="true" />
```

```
<field name="publishedDate" type="date" stored="true" indexed="true" />
<field name="updatedDate" type="date" stored="true" indexed="true" />
- <!--
 Custom fields created for indexing our OCR processed books
 -->
<field name="publisher" type="string" stored="true" indexed="false" />
<field name="publisherURL" type="string" stored="true" indexed="false" />
<field name="domain" type="string" stored="true" indexed="false" />
<field name="group" type="string" stored="true" indexed="true" />
- <!--
 Custom fields created for images and business data
 -->
<field name="catagory" type="string" indexed="true" />
<field name="address" type="text_general" indexed="true" stored="true" />
<field name="city" type="string" indexed="true" stored="true" />
- <!--
 fields for creativecommons plugin
 -->
<field name="cc" type="string" stored="true" indexed="true" multiValued="true" />
- <!--
 fields for tld plugin
 -->
<field name="tld" type="string" stored="false" indexed="false" />
<field name="_version_" type="long" indexed="true" stored="true" />
  </fields>
<uniqueKey>id</uniqueKey>
<defaultSearchField>content_urdu</defaultSearchField>
<solrQueryParser defaultOperator="OR" />
  </schema>
```

## References:


1. https://www.quora.com/Information-Retrieval-What-is-inverted-index
2. https://www.elastic.co/guide/en/elasticsearch/guide/current/inverted-index.html
3. http://bigdata-madesimple.com/top-50-open-source-web-crawlers-for-data-mining/
4. https://www.quora.com/What-is-the-best-open-source-web-crawler-and-why
5. http://nutch.apache.org/
6. http://hadoop.apache.org/
7. https://hbase.apache.org/book.html
8. https://en.wikipedia.org/wiki/Apache_HBase
9. https://github.com/renepickhardt/metalcon/wiki/simpleNutchSolrSetup
10. http://wiki.apache.org/nutch/Nutch2Crawling
11. http://searchcloudcomputing.techtarget.com/definition/Hadoop
12. http://www.webopedia.com/TERM/H/hadoop_distributed_file_system_hdfs.html
13. http://hadoopinrealworld.com/namenode-and-datanode/
14. https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html