Maulana Azad National Institute of Technology
BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROJECT

ON

Inverted Indexing Of Links And Keywords Using Hadoop Framework

SUBMITTED IN PARTIAL FULFILLMENT FOR THE DEGREE OF BACHELOR OF TECHNOLOGY

SUBMITTED BY:

Aman Gupta 141112216

G.Purushotham 141112274

P.Prabhu babu 141112277

Avijeet Maurya 141112279

UNDER THE GUIDENCE OF

Dr. Nilay Khare

SESSION 2016-17

Maulana Azad National Institute of Technology
BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that **Aman Gupta, G.Purushotham**, **P.Prabhu Babu** and **Avijeet Maurya** students of B.Tech 3rd Year (Computer Science & Engineering), have successfully completed their project "**INVERTED INDEXING OF LINKS AND KEYWORDS USING HADOOP FRAMEWORK**" in partial fulfillment of their minor project in Computer Science & Engineering.

Dr. Nilay Khare — (Project Guide)

Dr. Rajesh Wadhwani — (Project coordinator)

Maulana Azad National Institute of Technology BHOPAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, hereby, declare that the following report which is being presented in the Minor Project Documentation entitled "**INVERTED INDEXING OF LINKS AND KEYWORDS USING HADOOP FRAMEWORK**" is the partial fulfillment of the requirements of the third year (sixth semester) Minor Project in the field of Computer Science And Engineering. It is an authentic documentation of our own original work carried out under the able guidance of Dr. Nilay Khare. The work has been carried out entirely at Maulana Azad National Institute of Technology, Bhopal. The following project and its report, in part or whole, has not been presented or submitted by us for any purpose in any other institute or organization.

We, hereby, declare that the facts mentioned above are true to the best of our knowledge. In case of any unlikely discrepancy that may possibly occur, we will be the ones to take responsibility.

Aman Gupta 141112216
G.Purushotham 141112274
P.Prabhu babu 141112277
Avijeet Maurya 141112279

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected guide Dr. Nilay Khare, for his valuable help and guidance. We are thankful for the encouragement that he has given us in completing this project successfully. His rigorous evaluation and constructive criticism was of great assistance.

It is imperative for us to mention the fact that this minor project could not have been accomplished without the periodic suggestions and advice of our project co-ordinator Dr. Rajesh Wadhwani.

We are also grateful to our respected director Dr. Narendra S Chaudhary for permitting us to utilize all the necessary facilities of the college.

Needless to mention is the additional help and support extended by our respected HOD, Dr. R. K. Pateriya, in allowing us to use the departmental laboratories and other services.

We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind co-operation and help.

INDEX

Abstract.....6
Project Introduction.....7
WebCrawler.....8
Working Of Web Crawler.....9
Web Crawler Architecture.....
BloomFilter.....11
Introduction.....11
Algorithm description.....11
Bloomfilter size calculator.....12
Applications of bloomfilter.....12
Hadoop Framework.....13
Introduction.....13
HDFS.....13
Map Reduce Programming model.....14
Inverted Indexing.....15
Introduction.....15
Inverted Indexing Procedure.....15
Project Setup.....16
Hardware and Software.....16
Code.....16
Result.....17
References.....17
Appendix.....18
Codes.....18
WebCrawler.....18
Main.java.....18
NewInstance.java.....18
CrawlerEngine.java.....21
Parser.java.....26
LinkInvertedIndex.....30
LinkInvertedIndex.java.....30
LinkInvertedIndexMapper.java.....31
LinkInvertedIndexReducer.java.....32
KeyWordInvertedIndex.....32
KeyWordInvertedIndex.java.....32
KeyWordInvertedIndexMapper.java.....34
KeyWordInvertedIndexReducer.java.....35

Abstract

Inverted Indexing is one of task that is performed before page ranking. Today web contains the billions and trillions of documents. And to retrieve the document related to user's query requires a huge computation and the time slot in which this computation is to done is very small. Therefore, search engine uses the inverted index of keywords to documents for efficient retrieval of document in given time slot. After storing the inverted index of links it is possible to rank the documents so that it may help in getting the higher rank page and thus giving a better experience to user. And thus before going to page ranking inverted index of links and keywords is to be done.

This project first crawls the web and form the forward indexing of links and keywords in documents and then uses bloom filter to remove the stop words and then use map-reduce paradigm to convert forward indexing of links and keywords to inverted indexing. This report presents the architecture of web crawler used for crawling the web. And how bloom filter can be used to remove the stop words and size requirement for bloom filter. And how inverted indexing of link and keywords is formed using Hadoop framework.

Project Introduction

Today whatever the information we need is present on the internet. Be it educational, professional, entertainment or any other field. As a result of which the size of web has grown to tremendously large size. There are millions and trillions of web page and goes on increasing day by day. As the size of web has grown the problem for querying the web and retrieving the document related to query has become more complex. Modern day search engine uses the concept of page ranking for retrieving the document related to query. But before we go for page ranking there is one phase that every search engine goes for, that is inverted indexing. Every search engine of modern day stores the inverted index statically and updates it periodically so as to keep search engine updated otherwise it will give outdated results.

For forming the inverted index search engine requires the powerful web crawler so as to crawl the web. While crawling the web there occurs many problems like Storage, Space efficiency, time taken to crawl the web. Which also needed to dealt with. After the web crawler has done its job, the task to convert the forward indexed data to inverted index starts. But the problem here is data collected becomes so large that processing in single machine becomes time consuming. To Deal with that problem we needed the parallel processing framework called Hadoop. Processing of data and converting the forward indexed data to inverted indexed data becomes much easier and faster because of Hadoop framework. While creating the inverted indexing of keywords we need to remove the stop words for this purpose we utilized the probabilistic data structure called Bloom filter to test whether word is stop word or not.

So our project involves the following main theoretical aspect:

- WebCrawler
- Bloom Filter
- Hadoop Framework and Map Reduce Paradigm.
- Inverted Indexing

Details of these topics are presented next.

WebCrawler

The World Wide Web has grown from a few thousand pages in 1993 to more than two billion pages at present. Due to this explosion in size, web search engines are becoming increasingly important as the primary means of locating relevant information. Such search engines rely on massive collections of web pages that are acquired with the help of web crawlers, which traverse the web by following hyperlinks and storing downloaded pages in a large database that is later indexed for efficient execution of user queries. Many researchers have looked at web search technology over the last few years, including crawling strategies, storage, indexing, ranking techniques, and a significant amount of work on the structural analysis of the web and web graph.

Thus, highly efficient crawling systems are needed in order to download the hundreds of millions of web pages indexed by the major search engines. In fact, search engines compete against each other primarily based on the size and currency of their underlying database, in addition to the quality and response time of their ranking function.

While it is fairly easy to build a slow crawler that downloads a few pages per second for a short period of time, building a high-performance system that can download hundreds of millions of pages over several weeks presents a number of challenges in system design, I/O and network efficiency, and robustness and manageability.

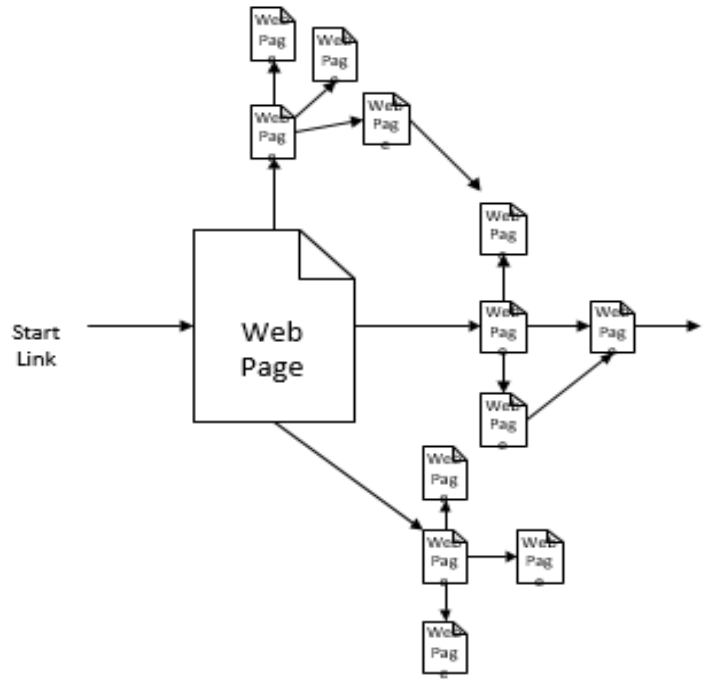


Figure – 1 Showing how crawler moves from one page to another

Working of web crawler

Our crawler starts with the link called start link or seed link. It gives the link to the parser. Parser then connect to the web and fetches the web page associated with the link. After getting the document parser starts its work on the document like parsing the web page for all the links that the web page contains and give it to link formatter which format all the links in a way so that we can store the data retrieved from the web efficiently (that is links can be flushed into the file at a time instead of flushing it one by one). Then web page is given to key word parser for parsing the keywords in the web page which extracts all the key words in the web page. Then all the key words are checked against the bloom filter to remove the stop words from it. During the extraction of words from the web page the frequency count of words is also done for further processing link document ranking.

Figure 1 shows how our web crawler starts from seed link and generates new links from the seed link and then it moves to the newly generated links to generate more links.

Types of Links:

(1) Inbound links or Inlinks

- Inbound links are links into the site from the outside.
- Inlinks are one way to increase a site's total Page Rank.
- Sites are not penalized for inlinks.

(2) Outbound links or Outlinks

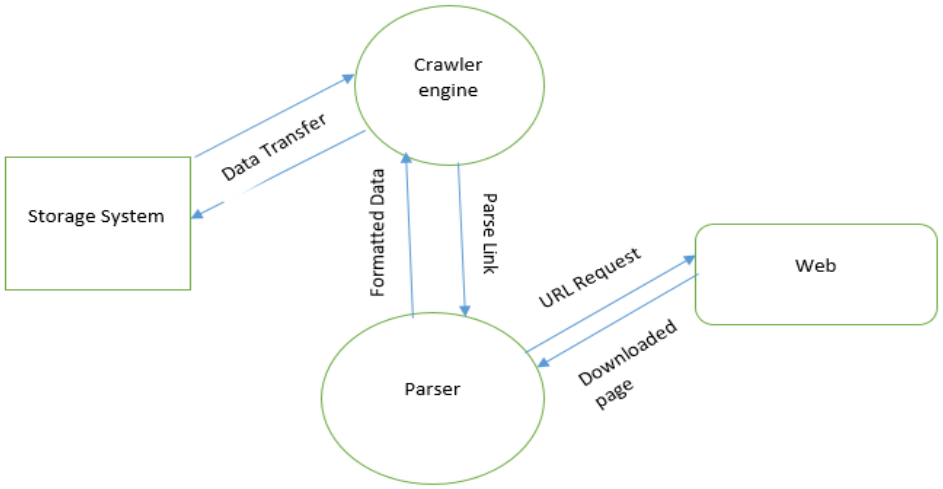
- Outbound links are links from a page to other pages in a site or other sites.

(3) Dangling links

- Dangling links are simply links that point to any page with no outgoing links.

Web Crawler Architecture in figure 2 and 3.

Figure - 2
Shows the flow of control from one class to another class.



Description of the architecture of our web crawler:

It is mainly divided into three components, CrawlerEngine, Parser and Storage System.

Storage System-:

After starting the web crawler, initial phase starts, which does the setup work like creating a directory for storing forward indexed data. It creates four files named queueFile for storing link that is to be crawled , crawledData file for storing forward indexing of links , crawledLink file for storing links that have been crawled successfully and keyWord file for storing forward indexing of keywords. After the work for initial phase is done the control is transferred to crawler engine.

CrawlerEngine-:

CrawlerEngine first takes data from Storage System and stores the data in main memory for fast access. Then it creates pool of threads and creates new thread if any thread dies. Each thread takes links from memory and passes it to parser. Each thread uses parser independently. After getting formated data from parser it stores it in storage system. The data structure used for storing the memory copy of queued links is a "SET" so that no dupilicate links are created. Set size can be changed depending upon ram size and no of threads can also be changed.

Parser-:

our parser takes a link from the crawler engine and connects our program to the web and fetches the required web page.After fetching the document,it extracts all the links and keywords in the page.All the keywords that are extracted will be passed through bloom filter for filtering out stop words .We used JSOUP library for extracting links and keywords .After extraction,parser formats all the links into a string format to store it in a efficient way.

.

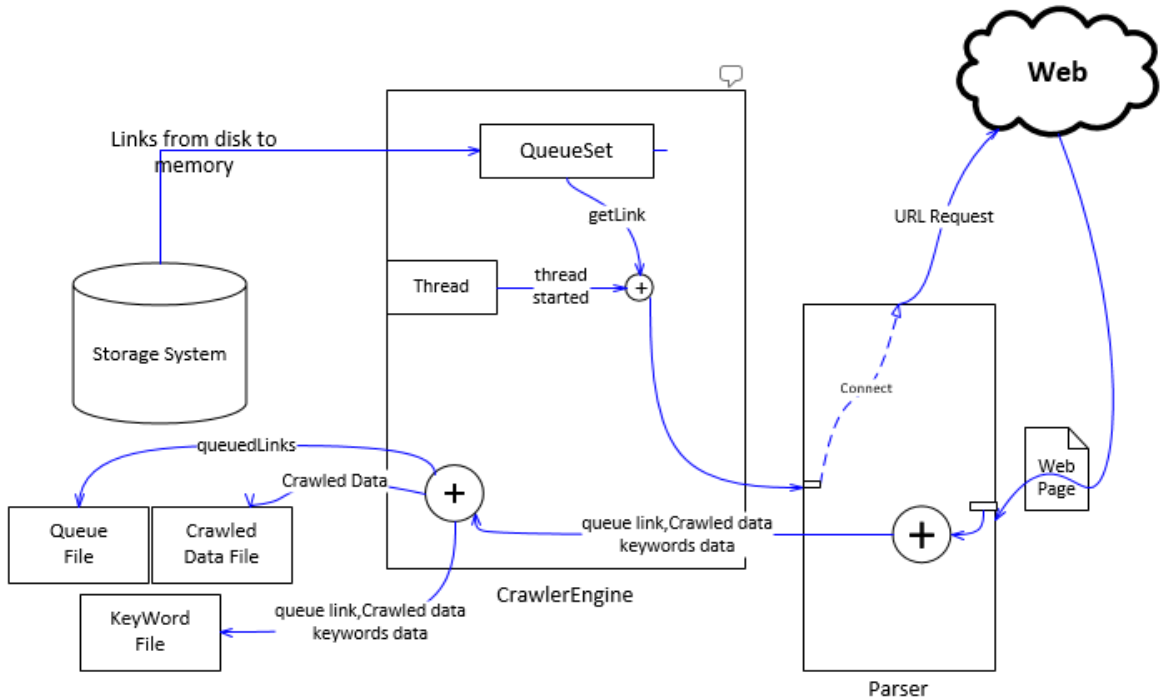


Figure – 3 core architecture of crawler. Showing control flow.

BloomFilter

Introduction

A Bloom filter is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not – in other words, a query returns either " possibly in set" or "definitely not in set". Elements can be added to the set, but not removed (though this can be addressed with a "counting" filter) the more elements that are added to the set, the larger the probability of false positives.

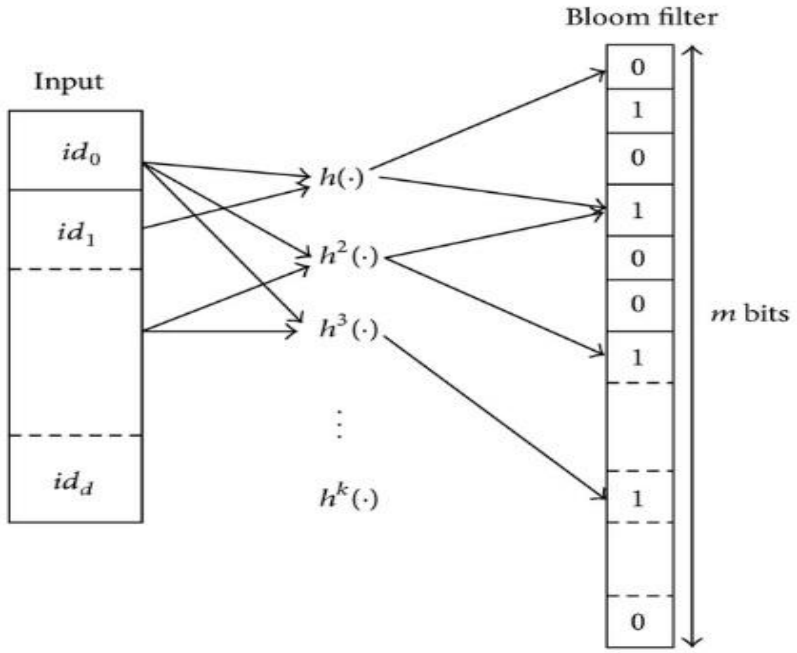


Figure - 4 Showing how Item inserted into bloom filter is hashed and bits in bit array of bloom filter is set.

Algorithm description

An empty Bloom filter is a bit array of m bits, all set to 0. There must also be k different hash functions defined, each of which maps or hashes some set element to one of the m array positions with a uniform random distribution. Typically, k is a constant, much smaller than m, which is proportional to the number of elements to be added; the precise choice of k and the constant of proportionality of m are determined by the intended false positive rate of the filter.

Adding items to bloom filter-:

To add an element, pass it to each of the k hash functions to get k array positions. Set the bits at all these positions to 1.

Membership test in bloom filter-:

To query for an element (test whether it is in the set), pass it to each of the k hash functions to get k array positions. If any of the bits at these positions is 0, the element is definitely not in the set – if it were, then all the bits would have been set to 1 when it was inserted. If all are 1, then either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive. In a simple Bloom filter, there is no way to distinguish between the two cases, but more advanced techniques can address this problem.

Bloomfilter size calculation

n - **N**umber of items in the filter

p - **P**robability of false positives, float between 0 and 1 or a number indicating 1-in-p

m **-** Nu**m**ber of bits in the filter

k **-** Number of hash functions

Formula for calculating size of bloom filter is:

```
m = ceil((n * log(p)) / log(1.0 / (pow(2.0, log(2.0)))));

k = round(log(2.0) * m / n);
```

Applications of bloom filter

1. The Google Chrome web browser used to use a Bloom filter to identify malicious URLs. Any URL was first checked against a local Bloom filter, and only if the Bloom filter returned a positive result was a full check of the URL performed (and the user warned, if that too returned a positive result).
2. The Squid Web Proxy Cache uses Bloom filters for cache digests.
3. Bitcoin uses Bloom filters to speed up wallet synchronization.

Hadoop Framework

Introduction

Apache Hadoop is an open-source software framework used for distributed storage and processing of big data sets using the MapReduce programming model. It consists of computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework.

The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model. Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers packaged code into nodes to process the data in parallel. This approach takes advantage of data locality, where nodes manipulate the data they have access to. This allows the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking.

The base Apache Hadoop framework is composed of the following modules:

Hadoop Common – contains libraries and utilities needed by other Hadoop modules.

Hadoop Distributed File System (HDFS) – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.

Hadoop YARN – a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications and

Hadoop MapReduce – an implementation of the MapReduce programming model for large scale data processing

HDFS

Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS).

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of

small computer machines in a reliable, fault-tolerant manner. HDFS uses a master/slave architecture where master consists of a single NameNode that manages the file system metadata and one or more slave DataNodes that store the actual data. A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode.

MapReduce Programming Model

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

Generally MapReduce paradigm is based on sending the computer to where the data resides! MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.

Map stage : The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.

Reduce stage : This stage is the combination of the Shuffle stage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.

The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.

After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

Inverted Index

Introduction

The inverted index data structure is a central component of a typical search engine indexing algorithm. A goal of a search engine implementation is to optimize the speed of the query: find the documents where word X occurs. Once a forward index is developed, which stores lists of words per document, it is next inverted to develop an inverted index. Querying the forward index would require sequential iteration through each document and to each word to verify a matching document. The time, memory, and processing resources to perform such a query are not always technically realistic. Instead of listing the words per document in the forward index, the inverted index data structure is developed which lists the documents per word.

With the inverted index created, the query can now be resolved by jumping to the word ID (via random access) in the inverted index.

Inverted Indexing Procedure

After collecting the data from web crawler in forward indexing format we process the data using map reduce. Inside the mapper program we take the forward indexing of page URL and reverse it in the form (target URL, source URL) and give it as output of mapper. Now the shuffle and sorting phase of framework comes into action and combines the output of mapper as (target URL , list(src1 URL, src2 URL…)). Which is the passed to the reducer which formats the outputs in required format to HDFS. Similar procedure is followed for keywords to document inverted indexing.

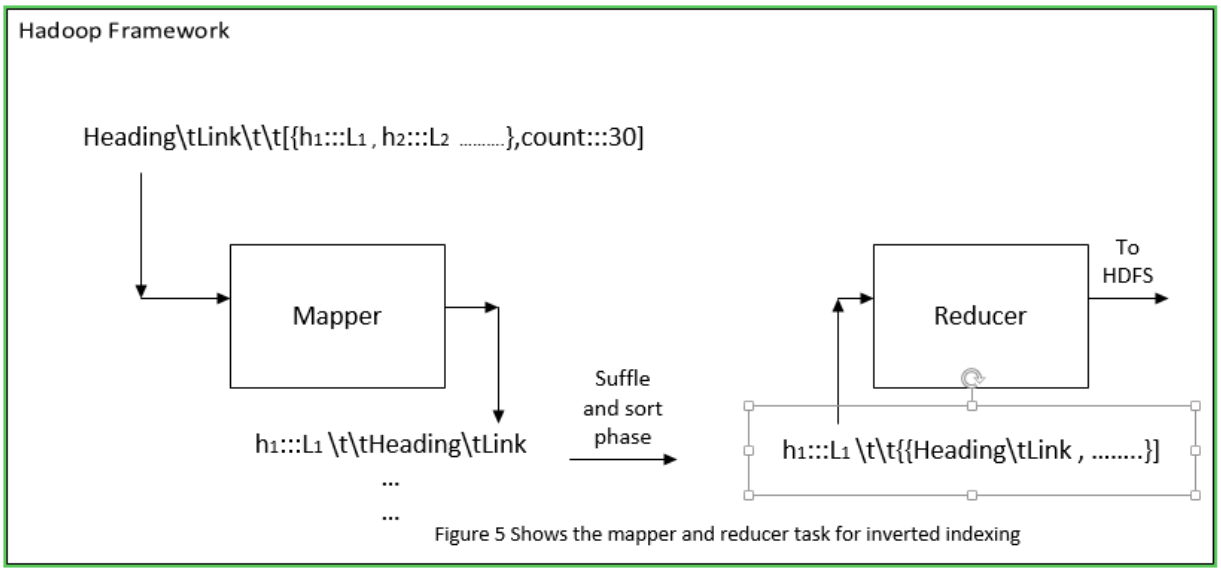


Figure 5 Shows the mapper and reducer task for inverted indexing

Project Setup

Hardware and Software

While running project following environment was used-

1. Eclipse.
2. Java
3. Windows 10 and Ubuntu in virtualbox.
4. Hadoop.
5. Intel Core-i3 processor machine

Codes

The codes implemented and used along with their descriptions are as follows-:

- Webcrawler:

  There are four java files (Main.java, NewInstance.java, CrawlerEngine.java, Parser.java). used for web crawling.

- LinkInvertedIndex:

  There are three java files (LinkInvertedIndex.java, LinkInvertedIndexMapper.java, LinkInvertedIndexReducer.java). used for converting forward index of links to inverted index.

- KeyWordInvertedIndex:

  There are three java files (KeyWordInvertedIndex.java, KeyWordInvertedIndexMapper.java. KeyWordInvertedIndexReducer.java). used for converting forward index of page and keyword to inverted index of keyword-page.

Results

Amount of Time Crawler runes	Links Crawled(no. of links)	Crawled data
5 min	**8551**	**208MB**
10 min	**27149**	**552MB**
15 min	**35704**	**772MB**

References

- http://ieeexplore.ieee.org/document/994750/authors

  V. Shkapenyuk CIS Dept., Polytech. Univ. Brooklyn, NY, USA

Appendix

Codes

A) WebCrawler

Main.java

```
import java.io.IOException;
public class Main {
     public static void main(String args[]) throws IOException{
          if(args.length<2){
               System.out.println("Provide two arguments <path to store
               links><starting link>");
               System.exit(1);
          }
          new NewInstance(args);
     }
}
```

NewInstance.java

```
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintStream;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
```

```
public class NewInstance {
	private String path;
	private String startLink;
	private String projectName;
	private String queueFile;
	private String crawlFile;
	private String crawledLinks;
	private String keyWordsFile;
	private String logFile;
	public NewInstance(String args[]) throws IOException{
		path = args[0]+"/WebCrawler";
		startLink = args[1];
		setupName();
		createDirectories();
		String queueFilePath = path+"/"+projectName+"/"+queueFile+".txt";
		String crawlFilePath = path+"/"+projectName+"/"+crawlFile+".txt";
		String crawledLinksPath = path+"/"+projectName+ "/" +
crawledLinks +".txt";
		String keyWordsPath = path+"/"+projectName+"/"+
keyWordsFile+".txt";
		new CrawlerEngine(queueFilePath,crawlFilePath,crawledLinksPath,
keyWordsPath);
	}
	public void setupName(){
		DateFormat df = new SimpleDateFormat("YYYY_MM_dd_hh_mm_ss");
		projectName = "WebCrawler_"+df.format(new Date());
		queueFile = "Queue";
		crawlFile = "Crawl";
		crawledLinks = "CrawledLinks";
		keyWordsFile = "KeyWords";
		logFile = "log";
	}
	public void createDirectories() throws IOException{
		if(Files.isDirectory(Paths.get(path))){
			createProjectDirectory();
			createQueueFile();
			createCrawlFile();
		}
		else{
			System.out.println(path +"Do Not Exists.");
			createProjectDirectory();
			createQueueFile();
			createCrawlFile();
		}
	}
```

```
	public void createQueueFile() throws IOException{
		try {
			Files.createFile(Paths.get(path+"/"+projectName+"/"+
	queueFile + ".txt"));
		} catch (IOException e) {
			e.printStackTrace();
			System.exit(1);
		}
		//code to insert first link in queue file
			BufferedWriter writer = new BufferedWriter(new
	OutputStreamWriter(new FileOutputStream(path+"/"+projectName+"/"+
		queueFile +".txt")));
			writer.write(startLink+"\n");
			writer.close();
	}
	public void createCrawlFile(){
		try {
			Files.createFile(Paths.get(path+"/"+projectName+"/"+
			crawlFile +".txt"));
		} catch (IOException e) {
			e.printStackTrace();
			System.exit(1);
		}
	}
	public void createProjectDirectory(){
		try {
			Files.createDirectories(Paths.get(path+"/"+projectName));
		} catch (IOException e) {
			e.printStackTrace();
			System.exit(1);
		}
	}
}
```

CrawlerEngine.java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.RandomAccessFile;
import java.math.BigInteger;
import java.util.HashSet;

import org.apache.hadoop.util.bloom.BloomFilter;
import org.apache.hadoop.util.bloom.Key;
import org.apache.hadoop.util.hash.Hash;

public class CrawlerEngine implements Runnable{
/*
 * Field member for storing the path
 * */
     private static String queueFilePath;
     private static String crawledFilePath;
     private static String crawledLinkPath;
     private static String keyWordsPath;
/*
 * Field member for storing set of links in memory
 * */
     private static HashSet<String> queueSet;
/*
 * Field member for storing queuedlinks in disk
 * */
     private static RandomAccessFile queueFile;
     private static BufferedWriter buff;
     private volatile boolean flag = false;
/*
 * Field member for storing configuration setting
 * */
     private final int SET_SIZE = 100000;//10000
     private final int NO_THREADS = 2000;//1000
     private static Thread[] threadPool;
```

```
/*
 * Field member for storing bloom filter
 * */
     private static BloomFilter stopWordBF;
/*
 * for changing channel position instead of shifting data from queue
file.
 */
     private static long position = 0;
     private static BigInteger Counter_Size = new BigInteger("4");
     private static BigInteger Counter_Current = new BigInteger("0");
     private static final int BUFFER_SIZE =  16384;//8192

     @SuppressWarnings("static-access")
     public CrawlerEngine(String queueFilePath,String crawledFilePath,
String crawledLinks,String keyWordsPath) throws IOException{
          this.queueFilePath = queueFilePath;
          this.crawledFilePath = crawledFilePath;
          this.crawledLinkPath = crawledLinks;
          this.keyWordsPath = keyWordsPath;
          queueSet = new HashSet<String>();
          buff = new BufferedWriter(new OutputStreamWriter(
new FileOutputStream(crawledLinkPath)));
          trainBloomFilter();
          Parser.initBloomFilter(stopWordBF);
          fill2QueueSet_ChangeWritePos();
//Run first Thread to crawl in parallel
          threadPool = new Thread[NO_THREADS];
          threadPool[0] = new Thread(this);
          threadPool[0].start();
          threadPool[0].setName("Thread 0");
//Wait till first Thread has done some work
          while(!flag){
               try {
                    Thread.sleep(1000);
               } catch (InterruptedException e) {
                    e.printStackTrace();
               }
          }
//Launch More Thread
          for(int i=1;i<NO_THREADS;++i){
               threadPool[i] = new Thread(this);
               threadPool[i].start();
               threadPool[i].setName("Thread "+i);
          }
```

```
//Test for thread to alive or not
          while(new File(queueFilePath).length() != 0){
               for(int i=0;i<NO_THREADS;++i){
                    if(!threadPool[i].isAlive()){
                         threadPool[i] = new Thread(this);
                         threadPool[i].setName("Thread "+i);
                         threadPool[i].start();
                    }
               }
               try {
                    Thread.sleep(60*1000);
               } catch (InterruptedException e) {
                    e.printStackTrace();
               }
          }
     }
//Work for each thread
     public void run() {
          System.out.println("Thread Started...");
          while(true){
               while(!queueSet.isEmpty()){
                    //get a link from set and give it to parser that will
                    return the scanned Link.
                    String link = "";
                    synchronized (this){
                         link = queueSet.iterator().next();
                         queueSet.remove(link);
                         System.out.println("|||||||QueueSet Size +"+
queueSet.size());
                    }
                    String[] generatedData = Parser.parser(link);
                    //code to store generated links
                    putLinksToQueueFile(generatedData[1]);
                    putGeneratedLinksToCrawledFile(generatedData[0]);
                    putKeyWordsToKeyWordsFile(generatedData[2]);
                    System.out.println("Link Added");
                    flag = true;
                    try {
                         buff.write(link+"\n");
                         buff.flush();
                    } catch (IOException e) {
                         e.printStackTrace();
                    }
               }
```

```
                synchronized (this){
                     if(new File(queueFilePath).length() == 0){break;}
                     else{
                       try {
                          if(Counter_Current.compareTo(Counter_Size)==-1){
                            fill2QueueSet_ChangeWritePos();
                               Counter_Current = Counter_Current.add(new
          BigInteger("1"));
                          }else{
                               Counter_Size = Counter_Size.multiply(new
          BigInteger("2"));
                               Counter_Current = new BigInteger("0");
                               QueueSet_ShiftData();
                               fill2QueueSet_ChangeWritePos();
                          }
                       }catch(IOException e) {
                          e.printStackTrace();
                       }
                     }
                }
          }
          System.out.println("Thread Finished...");
     }
//Training bloomfilter
     public void trainBloomFilter() throws IOException{
          BufferedReader buff = new BufferedReader(new InputStreamReader
(this.getClass().getClassLoader().getResourceAsStream("StopWord-
List.txt")));
          stopWordBF = new BloomFilter(47923,33,Hash.MURMUR_HASH);
          String line = "";
          while((line = buff.readLine())!= null){
               stopWordBF.add(new Key(line.getBytes()));
          }
          buff.close();
     }
     public synchronized void putLinksToQueueFile(String links){
          try {BufferedWriter queueFileWriter = new BufferedWriter(new
     OutputStreamWriter(new FileOutputStream(queueFilePath,true)));
               queueFileWriter.write(links);
               queueFileWriter.close();
          } catch (FileNotFoundException e) {
               e.printStackTrace();
          } catch (IOException e) {
               e.printStackTrace();
          }
     }
```

```
public synchronized void putGeneratedLinksToCrawledFile(String data){
          try {
               BufferedWriter crawledFileWriter = new BufferedWriter(new
     OutputStreamWriter(new FileOutputStream(crawledFilePath,true)));
               crawledFileWriter.write(data);
               crawledFileWriter.close();
          } catch (FileNotFoundException e) {
               e.printStackTrace();
          } catch (IOException e) {
               e.printStackTrace();
          }
     }
     public synchronized void putKeyWordsToKeyWordsFile(String
keyWords){
          try {
               BufferedWriter keyWordFileWriter = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream(keyWordsPath,true)));
               keyWordFileWriter.write(keyWords);
               keyWordFileWriter.close();
          } catch (FileNotFoundException e) {
               e.printStackTrace();
          } catch (IOException e) {
               e.printStackTrace();
          }
     }
     public synchronized void fill2QueueSet_ChangeWritePos() throws
     IOException{
          queueFile = new RandomAccessFile(queueFilePath,"rw");
          queueFile.seek(position);
          String line = "";
          int count = 0;
          while((count < SET_SIZE) && ((line = queueFile.readLine())
     != null)){
               queueSet.add(line.replace("\n", ""));
               position = queueFile.getFilePointer();
               count++;
          }
          if(count < SET_SIZE){
               queueFile.setLength(0);
               position = 0;
               System.out.println("Comes Here");
               return ;
          }
          queueFile.close();
     }
```

```
    public void QueueSet_ShiftData() throws IOException{
        long writePos = 0;
        long readPos = position;
        byte[] buffer = new byte[BUFFER_SIZE];
        int bytesRead;
        while(-1 != (bytesRead = queueFile.read(buffer))){
            queueFile.seek(writePos);
            queueFile.write(buffer,0,bytesRead);
            writePos += bytesRead;
            readPos += bytesRead;
            queueFile.seek(readPos);
        }
        queueFile.setLength(writePos);
        queueFile.close();
        position = 0;
    }
}
```

Parser.java

```
import java.io.IOException;
import java.net.SocketTimeoutException;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.UnknownHostException;
import java.util.HashMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.apache.hadoop.util.bloom.BloomFilter;
import org.apache.hadoop.util.bloom.Key;
import org.jsoup.HttpStatusException;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
```

```
public class Parser {
     private static BloomFilter stopWordBF;
     private static Pattern pattern = Pattern.compile("[a-zA-Z]*");
     public static void initBloomFilter(BloomFilter bloomFilter){
          stopWordBF = bloomFilter;
     }
     public static String[] collectLinkAndFormat(Elements links,
String url,String heading){
          StringBuilder crawled = new StringBuilder(heading+"\t"+url+
"\t\t[{");//crawled is for crawled data
          StringBuilder queued = new StringBuilder();
          int count = 0;
          for(Element link : links){
               String link_string = link.absUrl("href");
               String curr_heading = link.text();
               try {
                    if(new URI(link_string).getHost().startsWith
("en.wikipedia.org")){
                         if(link_string.contains("#")){
                              continue;
                         }
                         else if(link_string.endsWith(".png")||
                                   link_string.endsWith(".PNG")||
                                   link_string.endsWith(".jpg")||
                                   link_string.endsWith(".JPG")||
                                   link_string.endsWith(".exe")||
                                   link_string.endsWith(".EXE")||
                                   link_string.endsWith(".gif")||
                                   link_string.endsWith(".GIF")||
                                   link_string.endsWith(".pdf")||
                                   link_string.endsWith(".PDF")){
                              continue;
                         }
                         crawled.append(curr_heading+":::"+link_string
+" , ");
                         queued.append(link_string+"\n");
                         count++;
                    }
               }catch (URISyntaxException e) {
                    e.printStackTrace();
               }catch(Exception e){
                    System.out.println("Exception");
                    e.printStackTrace();
               }
          }
```

```
            crawled.delete(crawled.length()-2, crawled.length()-1);
            crawled.append("},count:"+count+"]\n");
            return new String[]{crawled.toString(),queued.toString()};
      }
      public static String[] parser(String url){
            try {
                  System.out.println("Inside Parser with "+url);
                  Document doc = Jsoup.connect(url).Timeout(20*1000)
.get();
                  String[] links = Parser.parseLink(url, doc);
                  String words = Parser.parseText2keyWords(url,doc);
                  return new String[]{links[0],links[1],words};
            }catch(NullPointerException e){
                  e.printStackTrace();
            }
            catch(SocketTimeoutException e){
                  Parser.parser(url);
            }
            catch(UnknownHostException e){
                  e.printStackTrace();
            }
            catch(HttpStatusException e){
                  e.printStackTrace();
            }
            catch (IOException e) {
                  Parser.parser(url);
            }
            return new String[]{"","",""};
      }
      public static String[] parseLink(String url,Document doc){
            Elements links = doc.getElementById("bodyContent").
      select("a[href]");
            String heading = doc.getElementById("firstHeading").text();
            return collectLinkAndFormat(links,url,heading);
      }
      public static String parseText2keyWords(String url,Document doc){
            String body = doc.getElementById("bodyContent").text();
            return collectKeyWordsAndFormat(url,body);
      }
      public static String collectKeyWordsAndFormat(String url,String
body){
            HashMap<String,Integer> keyWords = new HashMap<String,
      Integer>();
            Matcher matcher = pattern.matcher(body);
```

```
          while(matcher.find()){
               String key = body.substring(matcher.start(),matcher
.end()).toLowerCase();
                    if (key.length() > 0) {
                         if (!stopWordBF.membershipTest(new Key(key
.getBytes()))) {
                              if (keyWords.containsKey(key)) {
                                   keyWords.put(key, keyWords.get(key) +
1);
                              } else {
                                   keyWords.put(key, 1);
                              }
                         }
                    }
          }
          StringBuilder keyWordBuilder = new StringBuilder
     (url+"\t\t[{");
          for(String keyWord : keyWords.keySet()){
               keyWordBuilder.append(keyWord+":"+(int)keyWords.get(
     keyWord)+" , ");
          }
          keyWordBuilder.delete(keyWordBuilder.length()-2,
     keyWordBuilder.length()-1);
          return keyWordBuilder.append("}]\n").toString();
     }
}
```

B) LinkInvertedIndex

LinkInvertedIndex.java

```
package org.aman.hadoop;
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class LinkInvertedIndex {
     public static void main(String[] args) throws IOException,
ClassNotFoundException, InterruptedException{
          Configuration conf = new Configuration();
          String[] otherArgs = new GenericOptionsParser(conf,args)
     .getRemainingArgs();
          if(otherArgs.length < 2 ){
               System.out.println("Supply <inputlocation>
     <outputlocation");
               System.exit(1);
          }
          Job job = Job.getInstance(conf,"LinkInvertedIndex");
          job.setJarByClass(LinkInvertedIndex.class);
          job.setMapperClass(LinkInvertedIndexMapper.class);
          job.setReducerClass(LinkInvertedIndexReducer.class);

          job.setMapOutputKeyClass(Text.class);
          job.setMapOutputValueClass(Text.class);
          job.setOutputKeyClass(Text.class);
          job.setOutputValueClass(Text.class);
          FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
          FileOutputFormat.setOutputPath(job,new Path(otherArgs[1]));
          System.exit(job.waitForCompletion(true)?0:1);
     }
}
```

LinkInvertedIndexMapper.java

```
package org.aman.hadoop;
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class LinkInvertedIndexMapper extends Mapper
     <LongWritable,Text,Text,Text> {
     private Pattern pattern;
     public void setup(Context context) throws IOException,
      InterruptedException{
          pattern = Pattern.compile("\\{[^\\}]*\\}");
     }
     public void map(LongWritable key ,Text value,Context context)
throws IOException, InterruptedException{
               try {Text newKey, newValue;
                    String[] valueSplit = value.toString()
     .split("\t\t");
                    if(valueSplit.length == 2){
                         newValue = new Text(valueSplit[0].replace
     ("\t", ":::"));
                         Matcher matcher = pattern.matcher(
     valueSplit[1]);
                         while(matcher.find()){
                              StringBuilder valueStrBuilder = new
StringBuilder(valueSplit[1].substring(matcher.start()+1,matcher
     .end()-1));
                              valueSplit[1] = valueStrBuilder
     .toString();
                         }
                         for(String split : valueSplit[1].split(" , ")){
                              newKey = new Text(split);
                              context.write(newKey, newValue);
                         }
                    }
               } catch (StringIndexOutOfBoundsException e) {
                    e.printStackTrace();
               }
     }
}
```

LinkInvertedIndexReducer.java

```
package org.aman.hadoop;
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class LinkInvertedIndexReducer extends
     Reducer<Text,Text,Text,Text> {

     public void reduce(Text key ,Iterable<Text> values, Context
     context) throws IOException, InterruptedException{
          StringBuilder sb = new StringBuilder("[");
          for(Text value : values){
               sb.append(value.toString()+" , ");
          }
          sb.replace(sb.length()-2,sb.length(), "");
          sb.append("]");
          context.write(key, new Text(sb.toString()));
     }
}
```

C) KeyWordInvertedIndex

KeyWordInvertedIndex.java

```
package org.aman.hadoop;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class KeyWordInvertedIndex {
     public static void main(String[] args) throws IOException, Class-
NotFoundException, InterruptedException{
          Configuration conf = new Configuration();
          String[] otherArgs = new GenericOp-
tionsParser(conf,args).getRemainingArgs();
          if(otherArgs.length < 2 ){
               System.out.println("Supply <inputlocation> <outputloca-
tion");
               System.exit(1);
          }
          Job job = Job.getInstance(conf,"KeyWordInvertedIndex");
          job.setJarByClass(KeyWordInvertedIndex.class);
          job.setMapperClass(KeyWordInvertedIndexMapper.class);
          job.setReducerClass(KeyWordInvertedIndexReducer.class);

          job.setMapOutputKeyClass(Text.class);
          job.setMapOutputValueClass(Text.class);
          job.setOutputKeyClass(Text.class);
          job.setOutputValueClass(Text.class);

          FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
          FileOutputFormat.setOutputPath(job,new Path(otherArgs[1]));

          System.exit(job.waitForCompletion(true)?0:1);
     }
}
```

KeyWordInvertedIndexMapper.java

```
package org.aman.hadoop;
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class KeyWordInvertedIndexMapper extends Mapper<
LongWritable,Text,Text,Text>{
     private Pattern pattern;
     public void setup(Context context) throws IOException,
InterruptedException{
          pattern = Pattern.compile("\\{[^\\}]*\\}");
     }
     public void map(LongWritable key ,Text value,Context context)
throws IOException, InterruptedException{
          try {     Text newKey;
                    Text newValue;
                    String[] valueSplit =
value.toString().split("\t\t");
                    if(valueSplit.length == 2){
                         Matcher matcher = pattern.matcher(
valueSplit[1]);
                         while(matcher.find()){
                              StringBuilder valueStrBuilder = new
StringBuilder(valueSplit[1].substring
(matcher.start()+1,matcher.end()-1));
                              valueSplit[1] = valueStrBuilder
.toString();
                         }
                         for(String split : valueSplit[1].split(" ,
")){
                              String[] wordANDcount =
split.split(":");
                              newValue = new Text(valueSplit[0]+
":::"+wordANDcount[1]);
                              newKey = new Text(wordANDcount[0]);
                              context.write(newKey, newValue);
                         }
                    }
               } catch (Exception e) {
                    e.printStackTrace();
               }}
```

KeyWordInvertedIndexReducer.java

```
package org.aman.hadoop;
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class KeyWordInvertedIndexReducer extends Re-
ducer<Text,Text,Text,Text> {
     @Override
     public void reduce(Text key ,Iterable<Text> values, Context
context) throws IOException, InterruptedException{
          StringBuilder sb = null;
          try {
               sb = new StringBuilder("[{");
               for(Text value : values){
                    sb.append(value.toString()+" , ");
               }
               sb.replace(sb.length()-2,sb.length(), "");
               sb.append("}]");
               context.write(key,new Text(sb.toString()));
          } catch (Exception e) {
               e.printStackTrace();
          }

     }
}
```