School of Computing Science and Engineering

VIT University

Vandalur - Kelambakkam Road, Chennai - 600127

Programme: B.Tech - CSE

Course: CSE4020 – Large Scale Data Processing

Slot: G1

Faculty: Prof. Maheshwari .N

Semester:Winter - 18

Component: J

Topic: INVERTED INDEXING USING MAPREDUCE PROG.

Hadoop - Version 1.0

Prepared by:

MUKUL CHHIKARA – 15BCE1221

JAGPREET SINGH THAKUR -15BCE1361

VELLORE INSTITUTE OF TECHNOLOGY(CHENNAI CAMPUS)

DATE : 6/04/2018

**Contents:**

1. **INTRODUCTION :**

Inverted index is index data structure for storing mapping results from content, such as words or numbers, to its locations in a database file or in a document or a set of documents. Most of the text searching systems rely on inverted index to search the documents that contains a given word or a term.

1.1 PURPOSE

Map Reduce implements various mathematical algorithms to divide a task into small parts and assign them to multiple systems. In technical terms, Map Reduce algorithm helps in sending the Map & Reduce tasks to appropriate servers in a cluster.

Like:> Sorting ,Searching ,Indexing ,TF-IDF .

where we implemented few small things in indexing algorithm .

1.2 INDEXING

Normally indexing is used to point to a particular data and its address. It performs batch indexing on the input files for a particular Mapper.The indexing technique that is normally used in MapReduce is known as inverted index .  Search engines like Google and Bing use inverted indexing technique.

1. **Working :**

2.1. make simple (.txt) file and inside the file put some Entry in terms of document Id .and then give each doc different ID.

Example:

Doc[0] = "i am jagpreet singh thakur"

Doc[1] = "who is he"

Doc[2] = "he is mukul"

2.2. We put that file into Hadoop dir . // command -> ./Hadoop fs -put <host file path> <hadoop dir path>

2.3. After that goto eclipse and make new (.jar) File .

2.4. finally put aeverything with (.jar) file in to hdfs and run our programme // command Hadoop jar <.jar file path> Class Name <Hadoop dir path where input file is present> <Hadoop dir path where we should store our output>.

2.5. After applying the Indexing algorithm, we get Like OUTPUT(Demo)

Doc[1] Doc[2] Doc[0]

"he Doc[2]

"i Doc[0]

"who Doc[1]

am Doc[0]

he" Doc[1]

is Doc[1] Doc[2]

jagpreet Doc[0]

mukul" Doc[2]

singh Doc[0]

thakur" Doc[0]

Where "a": {3} implies the term "a" appears in the Doc[3] file. Similarly, "is": Doc[1] Doc[2] implies the term "is" appears in the files Doc[1] and Doc[2].

**CODE:**

import java.io.IOException;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.Job;

public class InvertedIndex {

public static void main(String[] args)

throws IOException, ClassNotFoundException, InterruptedException {

if (args.length != 2) {

System.err.println("ONE OF input path AND output path IS MISSING ");

System.exit(-1);

}

Job job = new Job();

job.setJarByClass(InvertedIndexJob.class);

job.setJobName("Inverted Index");

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.setMapperClass(InvertedIndexMapper.class);

job.setReducerClass(InvertedIndexReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.waitForCompletion(true);

}

}

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class InvertedIndexMapper extends Mapper<LongWritable, Text, Text, Text> {

private Text wordText = new Text();

private final static Text document = new Text();

protected void map(LongWritable key, Text value, Context context)

throws java.io.IOException, InterruptedException {

String[] line = value.toString().split("=");

String documentName = line[0];

document.set(documentName);

String textStr = line[1];

String[] wordArray = textStr.split(" ");

for(int i = 0; i < wordArray.length; i++) {

wordText.set(wordArray[i]);

context.write(wordText,document);

}

}

}

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class InvertedIndexReducer extends

Reducer<Text, Text, Text, Text> {

protected void reduce(Text key, Iterable<Text> values, Context context)

throws java.io.IOException, InterruptedException {

StringBuffer buffer = new StringBuffer();

for (Text value : values) {

if(buffer.length() != 0) {

buffer.append(" ");

}

buffer.append(value.toString());

}

Text documentList = new Text();

documentList.set(buffer.toString());

context.write(key, documentList);

}

}

1. **Applications:**

The inverted index [data structure](https://en.wikipedia.org/wiki/Data_structure" \o "Data structure) is a central component of a typical [search engine indexing algorithm](https://en.wikipedia.org/wiki/Index_(search_engine)" \o "Index (search engine)). 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](https://en.wikipedia.org/wiki/Search_engine_indexing" \l "The_forward_index" \o "Search engine indexing) 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](https://en.wikipedia.org/wiki/Random_access" \o "Random access)) in the inverted index.

In pre-computer times, [concordances](https://en.wikipedia.org/wiki/Concordance_(publishing)" \o "Concordance (publishing)) to important books were manually assembled. These were effectively inverted indexes with a small amount of accompanying commentary that required a tremendous amount of effort to produce.

In bioinformatics, inverted indexes are very important in the [sequence assembly](https://en.wikipedia.org/wiki/Sequence_assembly" \o "Sequence assembly) of short fragments of sequenced DNA. One way to find the source of a fragment is to search for it against a reference DNA sequence. A small number of mismatches (due to differences between the sequenced DNA and reference DNA, or errors) can be accounted for by dividing the fragment into smaller fragments—at least one subfragment is likely to match the reference DNA sequence. The matching requires constructing an inverted index of all substrings of a certain length from the reference DNA sequence. Since the human DNA contains more than 3 billion base pairs, and we need to store a DNA substring for every index and a 32-bit integer for index itself, the storage requirement for such an inverted index would probably be in the tens of gigabytes.