PDF text customization - hadoop

By

Yugan S (19BCE1072)

Dinesh D (19BCE1133)

Bharathraj R (19BCE1110)

A project report submitted to

Dr. G. Bharadwaja Kumar

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

in partial fulfilment of the requirements for the course of

CSE3025 – LARGE SCALE DATA PROCESSING

In

B. Tech. COMPUTER SCIENCE AND ENGINEERING

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

VELLORE INSTITUTE OF TECHNOLOGY

CHENNAI - 600127

June 2021

TABLE OF CONTENTS

ABSTRACT

Pdf to text conversion program using java in hadoop environment. This is a pdf parser which converts the text content of the pdf more features can be modified accordingly by customizing input format.

Introduction

InputFormat describes the input-specification for a Map-Reduce job.The Map-Reduce framework relies on the InputFormat of the job to:
Validate the input-specification of the job.Split-up the input file(s) into logical InputSplits, each of which is then assigned to an individual Mapper.Provide the RecordReader implementation to be used to glean input records from the logical InputSplit for processing by the Mapper.
The default behavior of file-based InputFormats, typically sub-classes of FileInputFormat, is to split the input into logical InputSplits based on the total size, in bytes, of the input files. However, the FileSystem block size of the input files is treated as an upper bound for input splits. A lower bound on the split size can be set via mapreduce.input.fileinputformat.split.minsize.

Clearly, logical splits based on input-size is insufficient for many applications since record boundaries are to be respected. In such cases, the application has to also implement a RecordReader on whom lies the responsibility to respect

record-boundaries and present a record-oriented view of the logical InputSplit to the individual task.

Hadoop supports processing of many different formats and types of data through InputFormat. The InputFormat of a Hadoop MapReduce computation generates the key-value pair inputs for the mappers by parsing the input data. InputFormat also performs the splitting of the input data into logical partitions, essentially determining the number of Map tasks of a MapReduce computation and indirectly deciding the execution location of the Map tasks. Hadoop generates a map task for each logical data partition and invokes the respective mappers with the key-value pairs of the logical splits as the input.

Objective

Implement custom InputFormat implementations to gain more control over the input data as well as to support proprietary or application specific input data file format as inputs to Hadoop Mapreduce computations.

Proposed System

One is a similar class like the default TextInputFormat.class for pdf. We can call it as PdfInputFormat.class.

Second one is a similar class like the default LineRecordReader for handling pdf. We can call it as PdfRecordReader.clas

In source code of hadoop, the default TextInputFormat class is extended from a parent class called FileInputFormat.So, create a PdfinputFormat class extending the FileInputFormat class.
This will contain a method called createRecordReader which it got from the parent class.
We are calling our custom PdfRecordReader class from this createRecordReader method.

Software Requirements

- Linux operating system
- Hadoop environment
- Eclipse
- PDFBox-jar

Hardware requirements

- Intel Core 2 Duo/Quad/hexa/Octa or higher end 64 bit processor PC or Laptop (Minimum operating frequency of 2.5GHz)
- Hard Disk capacity of 1- 4TB.
- 64-512 GB RAM.
- 10 Gigabit Ethernet or Bonded Gigabit Ethernet.

Summary

The PdfRecordReader is a custom class created by us extending the RecordReader.

This mainly contains five methods which are inherited from the parent RecordReader class.
Initialize(),nextKeyValue(),getCurrentKey(),getCurrentValue(),getProgress(), close().
The logic explained below:
We are applying our pdf parsing logic in this method. This method will get the input split and we parses the input split using our pdf parser logic. The output of the pdf parser will be a text which will be stored in a variable. Then we splits the text into multiple lines by using '/n' as the splitter and we will store this lines in an array.

We need to send this as a key-value pair. So send the line number as the key and each line as the value. So the logic for checking getting from the array, setting it as key and value.

CONCLUSION AND FUTURE ENHANCEMENT

Conclusion:

To find small data in a large pdf more efficiently this method can be applied to get appropriate results, also to compile a large pdf data to text format.So,it helps in converting pdfs on a large scale.

Future Scope:

Extract Text

Extract Unicode text from PDF files.

Split & Merge

Split a single PDF into many files or merge multiple PDF files.

Fill Forms

Extract data from PDF forms or fill a PDF form.

Preflight

Validate PDF files against the PDF/A-1b standard.

Print

Print a PDF file using the standard Java printing API.

Save as Image

Save PDFs as image files, such as PNG or JPEG.

Create PDFs

Create a PDF from scratch, with embedded fonts and images.

Signing

Digitally sign PDF files.

Appendix

PROGRAMMING:

The PdfInputFormat code:

```
package com.amal.pdf;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

public class PdfInputFormat extends FileInputFormat {

	@Override
	public RecordReader createRecordReader(
			InputSplit split, TaskAttemptContext context)
throws IOException,
			InterruptedException {

		return new PdfRecordReader();
	}

}
```

The PdfRecordReader code:

```
package com.amal.pdf;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.util.PDFTextStripper;

public class PdfRecordReader extends RecordReader {
```

```
        private String[] lines = null;
        private LongWritable key = null;
        private Text value = null;

        @Override
        public void initialize(InputSplit genericSplit, TaskAttemptContext
context)
                        throws IOException, InterruptedException {

                FileSplit split = (FileSplit) genericSplit;
                Configuration job = context.getConfiguration();
                final Path file = split.getPath();

                /*
                 * The below code contains the logic for opening the file and
seek to
                 * the start of the split. Here we are applying the Pdf Parsing
logic
                 */

                FileSystem fs = file.getFileSystem(job);
                FSDataInputStream fileIn = fs.open(split.getPath());
                PDDocument pdf = null;
                String parsedText = null;
                PDFTextStripper stripper;
                pdf = PDDocument.load(fileIn);
                stripper = new PDFTextStripper();
                parsedText = stripper.getText(pdf);
                this.lines = parsedText.split("\n");

        }

        @Override
        public boolean nextKeyValue() throws IOException, InterruptedException {

                if (key == null) {
                        key = new LongWritable();
                        key.set(1);
                        value = new Text();
                        value.set(lines[0]);
                } else {
                        int temp = (int) key.get();
                        if (temp < (lines.length - 1)) {
                                int count = (int) key.get();
                                value = new Text();
                                value.set(lines[count]);
                                count = count + 1;
                                key = new LongWritable(count);
                        } else {
                                return false;
                        }

                }
                if (key == null || value == null) {
                        return false;
                } else {
```

```
                        return true;
                }
        }

        @Override
        public LongWritable getCurrentKey() throws IOException,
                        InterruptedException {

                return key;
        }

        @Override
        public Text getCurrentValue() throws IOException, InterruptedException {

                return value;
        }

        @Override
        public float getProgress() throws IOException, InterruptedException {

                return 0;
        }

        @Override
        public void close() throws IOException {

        }

}
```

Driver Class:

```
package com.amal.pdf;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
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.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class PdfInputDriver {

        public static void main(String[] args) throws IOException,
                        InterruptedException, ClassNotFoundException {
                Configuration conf = new Configuration();
                GenericOptionsParser parser = new GenericOptionsParser(conf,
args);
                args = parser.getRemainingArgs();
```

```
                Job job = new Job(conf, "Pdfwordcount");
                job.setJarByClass(PdfInputDriver.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(LongWritable.class);
                job.setInputFormatClass(PdfInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);
                FileInputFormat.setInputPaths(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
                job.setMapperClass(WordCountMapper.class);
                job.setReducerClass(WordCountReducer.class);

                System.out.println(job.waitForCompletion(true));
        }
}
```

Mapper Class:

```
package com.amal.pdf;

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.TaskAttemptContext;

public class WordCountMapper extends
                Mapper {
        private Text word = new Text();
        private final static LongWritable one = new LongWritable(1);

        protected void map(LongWritable key, Text value, Context context)
                        throws IOException, InterruptedException {
                String line = value.toString();
                StringTokenizer tokenizer = new StringTokenizer(line);
                while (tokenizer.hasMoreTokens()) {
                        word.set(tokenizer.nextToken());
                        context.progress();
                        context.write(word, one);
                }
        }
}
```

Reducer Class:

```
package com.amal.pdf;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends
                Reducer {
```

```
        protected void reduce(Text key, Iterable values,
                        Context context) throws IOException,
InterruptedException {
                int sum = 0;
                for (LongWritable value : values) {
                        sum += value.get();

                }
                context.write(key, new LongWritable(sum));
        }
}
```

INPUT PDF FILE:

A Dust of Snow

By Robert Frost

The way a crow
Shook down on me
The dust of snow
From a hemlock tree

Has given my heart
A change of mood
And saved some part
Of a day I had rued.

OUTPUT SCREENSHOTS:

```
2021-05-30 12:57:28,675 INFO mapreduce.Job: Counters: 36
        File System Counters
                FILE: Number of bytes read=17108
                FILE: Number of bytes written=1248574
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=359760
                HDFS: Number of bytes written=202
                HDFS: Number of read operations=15
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Map input records=12
                Map output records=35
                Map output bytes=296
                Map output materialized bytes=372
                Input split bytes=105
                Combine input records=0
                Combine output records=0
                Reduce input groups=30
                Reduce shuffle bytes=372
                Reduce input records=35
                Reduce output records=30
                Spilled Records=70
                Shuffled Maps =1
```

File information - part-r-00000

Download Head the file (first 32K) Tail the file (last 32K)

Block information -- Block 0

Block ID: 1073741849

Block Pool ID: BP-241061458-127.0.1.1-1617877167133

Generation Stamp: 1025

Size: 202

Availability:

- ubuntu

File contents

```
A    2
And  1
By   1
Dust     1
From     1
Frost    1
Has  1
Robert   1
```

```
 1 A        2
 2 And      1
 3 By       1
 4 Dust     1
 5 From     1
 6 Frost    1
 7 Has      1
 8 Robert   1
 9 Shook    1
10 Snow     1
11 The      2
12 a        2
13 change   1
14 crow     1
15 down     1
16 dust     1
17 given    1
18 heart    1
19 hemlock          1
20 me       1
21 mood     1
22 my       1
23 of       3
24 on       1
25 part     1
26 saved    1
27 snow     1
28 some     1
29 tree     1
30 way      1
```

REFERENCES

- https://pdfbox.apache.org/
- https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html#Job+Input
- https://intellitech.pro/tutorial-4-hadoop-custom-input-format/