Task 1.1

a. Look at the contents of the folder “output” - what are the files placed in there? What do they mean?

Files: \_SUCCESS, part-r-00000

On the successful completion of a job, the MapReduce runtime creates a \_SUCCESS file in the output directory. This may be useful for applications that need to see if a result set is complete just by inspecting HDFS.

The output files are by default named part-x-yyyyy where: x is either 'm' or 'r', depending on whether the job was a map only job, or reduce. yyyyy is the mapper or reducer task number (zero based).

b. How many times did the word ‘Discovery’ (case-sensitive) appear in the text you analyzed?

Discovery: 5

c. In this example we used Hadoop in “Local (Standalone) Mode”. What is the difference between this mode and the Pseudo-distributed mode?

Standlone: This mode runs on a single machine, without a distributed file system, but directly reads and writes the file system of the local operating system

Pseudo-distributed: This mode also runs on a single machine, but uses different Java processes to simulate various nodes in distributed operation: (NameNode, DataNode, JobTracker, TaskTracker, SecondaryNameNode)

Task 1.2

a. What are the roles of the files core-site.xml and hdfs-site.xml ?

We configurate the global parameters of cluster including HDFS URL and temporary directory of Hadoop by editing core-site.xml. The parameters of HDFS are stored in hdfs-site.xml, such as the storage location of the name node and data node, the number of file copies, the read permission of the file, etc.

b. Describe briefly the roles of the different services listed when executing ‘jps’.

DataNode: it stores data block

SecondaryNameNode: it helps NameNode merge edits log to reduce NameNode’s startup time

NameNode: its job is to accept the read and write requests from client and send them to DataNode

Task 1.3

a. Explain the roles of the different classes in the file WordCount.java.

The map function is represented by the Mapper class, which declares the abstract map method.

The reduce function is defined by Reducer class.

b. What is HDFS, and how is it different from the local filesystem on your

virtual machine?

HDFS (Hadoop Distributed File System) is a special kind of distributed file system that is designed to be suitable for being used on low-cost hardware. It is high fault-tolerant and provides high throughput for applications to access the stored data, which make it to be suitable for applications with large data sets.

HDFS and local filesystem on virtual machine both consist of block, but they have different definition for block. On HDFS, block means the Linux file and its default smallest size is 64MB. While for local filesystem on Linux, the block is the block of physical disk block, which indicate that HDFS is composed of multiple Linux files and Linux local filesystem is composed of data blocks. What’s more, in HDFS, namenode records the datanode where the linux data file is located, but in Linux local filesystem, inode will record the block pointer of the data area where the file is stored.

Task 1.4

徽标

描述已自动生成

图表, 条形图

描述已自动生成

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

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

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

public class FirstLetterCount {

public static class FirstLetterCountMapper

extends Mapper<LongWritable, Text, Text, IntWritable> {

@Override

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

throws IOException, InterruptedException {

# change the input data into string and make all the letters of it be lowercase

String line = value.toString().toLowerCase();

# split the string data into individual word by blankspace

String[] strings = line. split(" ");

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

# filter out empty strings

if ( strings[i] != null && !strings[i].isEmpty() ){

# extract the first letter of this word

char firstLetter = strings[i].charAt(0);

# find the words begin with a-z and deliver their first letter to reducer class

if (firstLetter >= 'a' && firstLetter <= 'z') {

context.write(new Text(String.valueOf(firstLetter)), new IntWritable(1));

}

}

}

}

}

public static class FirstLetterCountReducer

extends Reducer<Text, IntWritable, Text, IntWritable> {

private IntWritable result = new IntWritable();

@Override

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

throws IOException, InterruptedException {

int sum = 0;

# count the number of words begin with a-z in a loop

for (IntWritable val : values) {

sum += val.get();

}

result.set(sum);

context.write(key, result);

}

}

public static void main(String[] args) throws Exception {

if (args.length != 2){

System.err.println("Err: CharacterCount requires <input path> <output path>");

System.exit(-1);

}

Configuration conf = new Configuration();

Job job = new Job(conf);

job.setJarByClassFirstLetterCount FirstLetterCount.class);

job.setJobName("FirstLetterCount Job");

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

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

job.setMapperClass(FirstLetterCountMapper.class);

job.setCombinerClass(FirstLetterCountReducer.class);

job.setReducerClass(FirstLetterCountReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

}

}

Task 1.5

1. One example of JSON formatted data is Twitter tweets:

https://dev.twitter.com/overview/api/tweets. Based on the twitter documentation,

how would you classify the JSON-formatted tweets structured, semi-structured

or unstructured data?

JSON-formatted tweets are semi-structured data. Because for semi-structured data, we can obtain corresponding information through flexible key value adjustment, and the data format is not fixed, the information stored under the same key value may be numeric, string, dictionary, or list. These meet the characteristics of JSON-formatted tweets data.

2. Elaborate on pros and cons for SQL and NoSQL solutions, respectively. Give

some examples of particular data sets/scenarios that might be suitable for these

types of databases. (expected answer length: 0.5 A4 pages)

SQL:

The biggest advantage of SQL is that it can keep the consistency of data. And then, the data schema is normalized so the cost of updating is low. What’s more, complex queries such as join are also avaliable in SQL, which improves the search speed greatly. As for the disadvantages, the first one is that it’s too expensive to maintain data consistency and indexes. Secondly, table structure of SQL is not easy to expand (stored data are all structured data). Last, when the amount of data is too large or the read and write operations are too frequent, the running speed will be much slower

NoSQL:

NoSQL allows flexible data model that it can be columnar storage, key-value storage and document storage. And because it has no schema and no static definition of how the data must be modeled, when requirements change, the entire schema does not have to be rewritten to better adapt to changing application needs. Unlike the traditional relational databases are limited by disk io, the pressure is doubled in the case of high concurrency, NoSQL databases like redis support 10w reads and writes per second. So even when we operate the large data set, NoSQL still can provide us high querying efficiency. One clear drawback of NoSQL is that only the relative consistency of the data can only be guaranteed, especially when the data is synchronized。 What’s more, the state of the master and slave servers is inconsistent and transactions are not supported by most NoSQL.

Since NoSQL databases do not have a rigid schema, they are easily scalable, flexible, and easy to use. They are ideal for applications that do not have a specific architectural definition, such as content management systems, big data applications, real-time analytics, etc. On the other hand, SQL databases have long been built with a fixed schema design and collection structure. They are ideal for applications that require multi-line transactions, such as accounting systems, or systems built for relational structures.

Task 2.1

Firstly, I create the default mapper.py and reducer.py, and to accomplish this task, I just need to adjust the code in mapper but keep the reducer to be the default version. I use the json package to load the tweets data and filter the retweet out. Then, I use regular expression to split each tweet into individual words and count the words that meet the requirement. After the edition of Python is finished, I use to jar command to run Hadoop by Python. After the counting data is obtained and saved in the output file part-00000, I copy this file from virtual machine to my local filesystem and then write the python code to visualize this data.

文本

低可信度描述已自动生成

图表, 条形图

描述已自动生成

Mapper:

import sys

import json

import re

pronouns = ["han", "hon", "den", "det", "denna", "denne", "hen", "unique\_tweet"]

# input comes from STDIN (standard input)

for line in sys.stdin:

# remove leading and trailing whitespace

line = line.strip()

if len(line) != 0:

# use json to load the tweets

jsonData = json.loads(line)

# determine whether it's retweet or not

if 'retweeted\_status' not in jsonData:

tweets = jsonData['text']

# split the text into words

pattern = re.compile(r"\w+")

words = list(map(str, pattern.findall(tweets)))

# increase counters

words.append('unique\_tweet')

for word in words:

word = word.lower()

if word in pronouns:

# write the results to STDOUT (standard output);

# what we output here will be the input for the

# Reduce step, i.e. the input for reducer.py

#

# tab-delimited; the trivial word count is 1

print(word, 1)

reducer:

from operator import itemgetter

import sys

current\_word = None

current\_count = 0

word = None

# input comes from STDIN

for line in sys.stdin:

# remove leading and trailing whitespace

line = line.strip()

# parse the input we got from mapper.py

word, count = line.split('\t', 1)

# convert count (currently a string) to int

try:

count = int(count)

except ValueError:

# count was not a number, so silently

# ignore/discard this line

continue

# this IF-switch only works because Hadoop sorts map output

# by key (here: word) before it is passed to the reducer

if current\_word == word:

current\_count += count

else:

if current\_word:

# write result to STDOUT

print(current\_word, current\_count)

current\_count = count

current\_word = word

# do not forget to output the last word if needed!

if current\_word == word:

print(current\_word, current\_count)

Task 2.2

I use Python to operate MongoDB, and the tweets data is unzipped and put in the same path of this Python file. In Python, I write four functions: data loading, data counting, plot drawing and main function. In main function, I use pymongo to connect MongoDB, create the database and collection，call other three functions to show the result. In the function of load\_data, I use some packages to make the application reads all tweets txt files and writes them into databse. In the function of count\_data, I use aggregate to select the words that meet the requirement and count their occurrence, then the result is stored on a dictionary. In the function of draw\_data, I use the package of seaborn to visualize the data saved in the dictionary.

Compare with using Hadoop, I can use the query of MongoDB to select the data I want directly when I use pymongo. However, the time for MongoDB to process data is much longer than Hadoop.



图表, 条形图

描述已自动生成

from importlib.resources import path

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

import pymongo

import json

import os

def load\_data(mycol):

path = "tweets"

files = os.listdir(path)

for fine in files:

if not os.path.isdir(fine):

f = open(path+"/"+fine)

for line in f:

line = line.strip()

if len(line) != 0:

jsonData = json.loads(line)

if 'retweeted\_status' not in jsonData:

x = mycol.insert\_one(jsonData)

#print(x)

return x

def count\_data(mycol):

mydict = {}

pipeline = [

{"$project": {"words": {"$regexFindAll": {"input": "$text", "regex": "(\*UCP)\\b(han|hon|den|det|denna|denne|hen)\\b", "options": "i"}}}},

{"$set": {"words": "$words.match"}},

{"$unwind": {"path": "$words", "preserveNullAndEmptyArrays": False}},

{"$group": {"\_id": {"$toLower": "$words"}, "count": {"$sum": 1}}}

]

result =mycol.aggregate(pipeline)

# put the counting result into dictionary

for i in result:

mydict[i["\_id"]] = i["count"]

mydict["unique tweets"] = mycol.count\_documents({})

print(mydict)

return mydict

def draw\_data(mycol, mydict):

pronouns = ['den', 'denna', 'denne', 'det', 'han', 'hen', 'hon']

unique\_value = mycol.count\_documents({})

mydict = {'den': 1324057, 'denna': 22716, 'denne': 4015, 'det': 532906, 'han': 778945, 'hen': 34419, 'hon': 363764}

for x in pronouns:

mydict[x] = mydict[x] / unique\_value

df = pd.DataFrame({"pronouns":list(mydict.keys()), "normalized number":list(mydict.values())})

sns.barplot(x="normalized number", y="pronouns", data=df)

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

# connect to MongoDB

myclient = pymongo.MongoClient('mongodb://localhost:27017/')

dblist = myclient.list\_database\_names()

mydb = myclient["tweets"]

mycol = mydb["tweet"]

load\_data(mycol)

a = count\_data(mycol)

draw\_data(mycol, a)