Kristiania
University
College

Course Assignment

Master of

Applied Computer Science

Department of Technology

Assignment title	Exploring Bitcoin
Course code	MA120
Course name	Big Data
Due date	18 Oktober

Declaration:

Through the submission of this assignment, we hereby declare that this report is the result of our own work, and that all sources have been properly cited to throughout the text.

Names of students	Theodore Midtbø Alstad ; Howie Chen
Student ID numbers	865317 ; 866354

Exploring Bitcoin

ABSTRACT

In this assignment we parsed a dataset, in the form of XMLs, through a Hadoop- and Pig apache. We learned how to write MapReduce jobs and Pig scripts, and how effective it is on our dataset based on XML files from Bitcoin stack exchange.

Introduction

We choose to work together because both of us have python background, therefor we choose python as main programming language. We explored Bitcoin as given dataset from archive.org/download/stackexchange/bitcoin.stackexchange.com.7z through Apaches: Hadoop and Pig.

Main functions

The datasets contains only XML files, which it means values to different attributes have a lot of ascii characters, punctiations, numbers and HTML tags. We also needed a function to parse the XML files to Hadoop and to create a mapper for the data. Therefor we created three different main functions thats being reused through the project. :

CleanBody

In cleanBody function it formats strings for mapper function. The function makes non-case sensitivity, removes ascii characters , HTML formatting, and treats anything saperated by blanked space or as separated words. Most of the words after separating/splitting will be counted as separate words, and this will effect the results of some tasks. For instance the name "Jens-Petter" will be counted as two words; "Jens" and "Petter". The input for the function is a string body which will be formatted and it returns a list of formatted string.

```
def cleanBody(body):
    body = body.lower()
    body = ascii(body)
    body = sub("<.+?>","",body)
    body = body.replace("/", " ")
    body = body.strip()

    for i in ignore_char:
          body = body.replace(i, "")

    body = body.split(" ")

    return body
```

Listing 1. CleanBody function

Mapper Core

mapper_core is the core of the mapper function; it prints out the relevant data in a format parseable by Hadoop. It functions through three modes, as determined by the parameter mode: "single", "double", and "triple".

- Single: Assumes input "words" is a list of words to print. Prints word in words as (word, 1). Ignores empty strings and spaces.
- Double: Assumes input "words" is two nested lists to print. Prints word, count in words as (word, count). Ignores empty strings and spaces.
- Triple: Assumes input "words" is three nested lists to print. Prints id, score, title in words as (id, score, title). does not ignore empty strings and spaces.

```
def mapper_core(words, mode="single"):
    if mode == "single":
        for word in words:
            if word not in ["", " "]:
              print("%s %s" %(word,1)) #Emit the word

    elif mode == "double":
        in1, in2 = words
        for word, count in zip(in1,in2):
            if word not in ["", " "]:
                print("%s %s" %(word,count)) #emit the words

    elif mode == "triple":
        in1, in2,in3  = words
        for id, score, title in zip(in1,in2,in3):
            print("%s %s %s" %(id,score, title)) #emit the words
```

Listing 2. mapper_core function

Xmlparser
The chosen method to interpret the dataset is to parse to the mapper function. Although parsing an entire XML-file takes up significant memory, this method fits our dataset. It has been separated out as a function so it may be easily replaced by other methods more fit for large files. The input for this function is a XML-file and the output is a parsed XML-file

```
def xmlparser(infile):
    if not isinstance(infile, str):
        infile = infile.detach()
    mytree = ET.parse(infile)
    myroot = mytree.getroot()
    return myroot
```

Listing 3. xmlparser function

Task 1 Warmup

1a) WordCount

Assumption: Count the words in body of questions PostTypeID="1" in the *Posts.xml* file. The result should be how many times a word occur in the body of questions.

ImplementationThe will

Notes/Reflection bye bye

```
abovennthanks 1
abovennthanksn 2
abovennwhat 1
abovennwill 1
aboventhe 1
aboventrying 1
abr 1
abra 4
abras 1
abreast 1
abridge 1
```

1a_output

1b) Unique words

Assumption: Write a MapReduce job where the result should be unique words in the titles PostTypeID="1" in the *Posts.xml* File.

Implementation

Notes/Reflection

```
asm
asn
aspect
aspects
aspnet
assemble
assembled
assembles
assembling
assert
assertion
```

1b_output

1c) MoreThan10

Assumption: Write a simple python code to check the title length in *Posts.xml*. The result should output how many titles have more than 10 words.

Implementation

Notes/Reflection

```
7600
```

1c_output

1d) Stopwords

Assumption: Write a simple python code based on task 1a to exclude stopwords from body of questions PostTypeID="1" in the *Posts.xml*. The output should be text file without any stopwords.

Implementation

Notes/Reflection

```
ati 8
atiradeon 1
atlcoin 1
atm 18
atms 5
atom 1
atomic 17
atomically 1
attach 8
attached 6
attaching 2
```

1d_output

1e) Pig top 10

Assumption: Write a pig script to select top 10 listed words after removing the stopwords from *Posts.xml*. The output should print out top 10 listed words and the corresponding occurrence rate.

Implementation

Notes/Reflection

```
bitcoin 5918
transaction 2376
wallet 2320
address 1550
block 1222
mining 1172
blockchain 1169
the 1148
```

```
transactions 1127
bitcoins 1050
```

1e_output

1f) Tags

Assumption: Write a MapReduce job to create a dictionary over unique tags in *Posts.xml*. The result should print the unique tags.

Implementation

Notes/Reflection

```
moneylaundering
moneysupply
moneytransfer
movecmd
msigna
mtgox
multibit
multibithd
multifactor
multigateway
multipartycomputation
```

1f_output

Task 2 Discover

2a) Counting

Assumption: We chose to write a MapReduce job to count the total unique users there are in *Users.xml*. The result will print out how many unique users there are.

Implementation

Notes/Reflection

```
56451
```

2a_output

2b) Unique users

Assumption: Write a MapReduce job based on task 2a) to create a mapper and a reducer functions in *Users.xml*. The result should contain unique users in the dataset.

Implementation

Notes/Reflection

```
10529
1053
10530
10531
10533
10534
10535
10536
10537
10538
10539
```

2b_output

2c) Top miners

Assumption:Write a MapReduce job to find top 10 users based on attribute Reputation="x" in *Users.xml*. The result will print out top 10 users based on their reputation.

Implementation

Notes/Reflection

```
david 87354
user 64135
pieter 51598
andrew 46149
murch 36524
thepiachu 31489
nick 29072
stephen 27509
chris 24736
nate 21620
```

2c_output

2d Top) questions

Assumption: Write a MapReduce job to find top 10 title questions PostTypeID="1" based on attribute Score="x" in *Posts.xml*.The result lists top 10 questions using id, question and the score.

Implementation

Notes/Reflection

2e) Favorite questions

Assumption: Write a MapReduce job to find top 10 title questions PostTypeID="1" based on attribute FavoriteCount="x" in *Posts.xml*.The result lists top 10 questions like id, question and the score.

Implementation

Notes/Reflection

2f) Average answers

Assumption:

Implementation

Notes/Reflection

2g) Countries

Assumption: We chose to write a MapReduce job to discover users by countries in *Users.xml*. The result lists different countries and corresponding users.

Implementation

Notes/Reflection

2h) Names

AssumptionWe chose to write a MapReduce job to find the most popular names in *Users.xml*. The result lists top 10 common names and how many.

Implementation

Notes/Reflection

2i) Answers

Assumption: Write a simple python code to find how many titles of questions PostTypeId="x" have at least one answers based on attribute AnswerCount in *Users.xml*. The result prints out how many questions have been answered.

Implementation

Notes/Reflection

Task 3 Numbers

3a) Bigram

Assumption: We chose to write a MapReduce job to find the most common pair of adjacent words in *Posts.xml*.For instance, "big data" or "Fast car" are examples of bigram. The result print the most common bigram

Implementation

Notes/Reflection

3b) Trigram

Assumption: This task is based on 3a, to find three words that appear consecutively in *Posts.xml*. The result print the most common trigram.

Implementation

Notes/Reflection

3c) Combiner

Assumption: Write a MapReduce job and add a combiner before the data sent to the reducer. The result should be the same as a reducer, but the reducer recive a smaller volume of data.

Implementation

Notes/Reflection Having a combiner in A MapReduce jobs saves us bandwidth and computational strain by decreasing the volume of data sent from the mapper. What combiner means is to have a semi-reducer after the mapper before sending it to the reducer.

3d) Useless

Assumption: We chose write a MapReduce job to find how many times the word "useless" in *Posts.xml* occurs in the body of questions PostTypeId="1". The result print how many times the word useless occurs.

Implementation

Notes/Reflection

Task 4 Search engine

4a) Title index

Assumption: We chose to write MapReduce job to create index over titles, bodies and answers of questions in *Posts.xml*. We are after having a simple index that lists publications in which a search term/s and occur/s. The result is a list of words and their index appearance in posts

Implementation

Notes/Reflection

```
angel, ,11695,8519,5197
angeles, ,5197,11695
angellist, ,11695
angels, ,11695,687
anger, ,2749
angered, ,11912
angle, ,2937,4110,1224
angles, ,1859
angry, ,2240,10044
```

4a_output

Conclusion