Objectives

1. Gain in depth experience playing around with big data tools (MapReduce, Hive and Spark).
2. Solve challenging big data processing tasks by finding highly efficient solutions.
3. Experience processing three different types of real data
    a. Standard multi-attribute data (Bank data)
    b. Time series data (Twitter feed data)
    c. Bag of words data.
4. Practice using programming APIs to find the best API calls to solve your problem. Here are the API descriptions for MapReduce, Hive and Spark (especially spark look under RDD. There are a lot of really useful API calls).

    a) [MapReduce] https://hadoop.apache.org/docs/r2.3.0/api/org/apache/hadoop/mapreduce/package-summary.html
    b) [Hive] https://cwiki.apache.org/confluence/display/Hive/LanguageManual
    c) [Spark] http://spark.apache.org/docs/1.3.0/api/scala/index.html#org.apache.spark.package
    - If you are not sure what a spark API call does, try to write a small example and try it in the spark shell

Expected quality of solutions

a) In general, writing more efficient code (less reading/writing from/into HDFS and less data shuffles) will be rewarded with more marks.
b) All MapReduce code you submit must be able to be compiled using the command "`javac -classpath `hadoop classpath` <code_files>`" on the Cloudera VM you received from us without requiring the installation of additional components.
c) All MapReduce code you submit should be runnable using "`hadoop jar <jar_uri> <hdfs_input_file> <hdfs_output_directory>`". For task 2C you need to allow the user to specify another two parameters being the x and y months respectively.
d) Using multiple MapReduce phases maybe appropriate for some of the subtasks. However, if you utilize multiple phases to solve a task, maintain a meaningful and logically consistent naming scheme for your files. (e.g.: Phase1.java, Phase2.java, ...)
e) For hive and spark code submissions, ensure that all commands relevant to accomplish the sub-task (i.e. 'create table' (hive), loading data AND queries!) are in the same file.
f) Scalability of the code is very important. This is especially important in terms of memory requirements of the mappers and reducers. For example writing a mapper that outputs the same key for any input, will result in all

the data going to a single reducer (no matter how many reducers you set). For example, if your mapper takes any string as input and always outputs the same **key** abc. This effectively means you will end up writing a sequential program. This is completely unacceptable and will result in **zero marks** for that subtask.

g) This entire assignment can be done using the Cloudera virtual machines supplied in the labs and the supplied data sets **without running out of memory**. Note task 3 is especially hard to do without running out of memory. But it is possible since we had done it. So it is time to show your skills!
h) Using combiners or local aggregation (inside the mapper) for MapReduce tasks where appropriate will be rewarded with marks. We will be looking at the total amount of data shuffled and awarding higher marks to lower amount of data shuffled.
i) Where ever appropriate use the fact the data is sorted according to intermediate key to reduce the work of the mapper and/or reducer.
j) I am not too fussed about the layout of the output. As long as it looks similar to the example outputs for each task. That will be good enough. The idea is not to spend too much time massaging the output to be the right format but instead to spend the time to solve problems.
k) For Hive queries. We prefer answers that use less tables.

Do the entire assignment using the Cloudera VM. Do not use AWS.

Tips:

1. Look at the data files **before** you begin each task. Try to understand what you are dealing with! You may find the shell commands "cat" and "head" helpful.

2. For each subtask we give very small example input and the corresponding output in the assignment specifications below. You should create input files that contain the same data as the example input and then see if your solution generates the same output.

3. In addition to testing the correctness of your code using the very small example input. You should also use the large input files that we provide to test the scalability of your solutions.

Task 1: Analysing Bank Data [35 marks total]

We will be doing some analytics on real data from a Portuguese banking institution. The data is related to their marketing campaign.

The data set used for this task can be found inside the bank directory of the assignment_datafiles.zip on LMS.

We got the data from the following source:
http://archive.ics.uci.edu/ml/datasets/Bank+Marketing

The data has the following attributes

Attribute number	Attribute name	Description
1	age	numeric
2	job	type of job (categorical: "admin.", "unknown", "unemployed", "management", "housemaid", "entrepreneur", "student", "blue-collar", "self-employed", "retired", "technician", "services")
3	marital	marital status (categorical: "married", "divorced", "single"; note: "divorced" means divorced or widowed)
4	education	(categorical: "unknown", "secondary", "primary", "tertiary")
5	default	has credit in default? (binary: "yes", "no")
6	balance	average yearly balance, in euros (numeric)
7	housing	has housing loan? (binary: "yes", "no")
8	loan	has personal loan? (binary: "yes", "no")
9	contact	contact communication type (categorical: "unknown", "telephone", "cellular")
10	day	last contact day of the month (numeric)
11	month	last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")
12	duration	last contact duration, in seconds (numeric)
13	campaign	number of contacts performed during this campaign and for this client (numeric, includes last contact)
14	pdays	number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)
15	previous	number of contacts performed before this campaign and for this client (numeric)
16	poutcome	outcome of the previous marketing campaign (categorical: "unknown","other","failure","success")
17	Term deposit	has the client subscribed a term deposit? (binary: "yes","no")

Here is a small example of the bank data that we will use to illustrate the subtasks below (we only list a subset of the attributes in this example, see the above table for the description of the attributes):

job	marital	education	balance	loan
management	Married	tertiary	2143	Yes
technician	Divorced	secondary	29	Yes
entrepreneur	Single	secondary	2	No
blue-collar	Married	unknown	1506	No
services	Divorced	secondary	829	Yes
technician	Married	tertiary	929	Yes
Management	Divorced	tertiary	22	No
technician	Married	primary	10	No

Using the entire bank data set downloaded from LMS please perform the following tasks. Please note we specify whether you should use [MapReduce] or [Hive] for each subtask at the beginning of each subtask.

a) [MapReduce] Report the number of clients of each job category. For the above small example data set you would report the following (output order is not important for this question):

management 2
technician 3
blue-collar 1
services 1
entrepreneur 1

[6 marks]

b) [Hive] Report the rounded average yearly income for all people in each education category. For the small example data set you would report the following (output order is not important for this question):

tertiary 1031
secondary 287
primary 10
unknown 1506

[6 marks]

c) [Hive] For each marital status report the percentage of people who have a personal loan. Hint you may need to use multiple queries or subqueries. For the small example data set you would report the following (output order is not important for this question):

Married 50%
Divorced 67%
Single 0%

[6 marks]

d) [MapReduce] Group balance into the following three categories:
   a. Low -infinity to 500
   b. Medium 501 to 1500
   c. High 1501 to +infinity

Report the number of people in each of the above categories. For the small example data set you would report the following (output order is not important in this question):

Low 4
Medium 2
High 2

[7 marks]

e) [MapReduce] For each education category report a list of people in descending order of balance. For each person report the following attribute values: *education category, balance, job, marital, loan*. Note this subtask can be done using a single or multiple MapReduce tasks. For the small example data set you would report the following (output order for education does not matter but order does matter for the attribute balance):

primary, 10, technician, married, no

secondary, 829, services, divorced, yes
secondary, 29, technician, divorced, yes
secondary, 2, entrepreneur, single, no

tertiary, 2143, management, married, yes
tertiary, 929, technician, married, yes
tertiary, 22, management, divorced, no

unknown, 1506, blue-collar, married, no

[10 marks]

Task 2: Analysing Twitter Time Series Data [30 marks]

In this task we will be doing some analytics on real twitter data. The data is a section of the data from obtained from the infochimps.org web site.

The data set used for this task can be found inside the twitter directory of the assignment_datafiles.zip on LMS. Note the data file is tab (\t) delimited.

The data is obtained from here:

http://www.infochimps.com/datasets/twitter-census-conversation-metrics-one-year-of-urls-hashtags-sm

The data has the following attributes

Attribute number	Attribute name	Description
1	Token type	In our data set all rows have Token type of hashtag. So this attribute is useless for this assignment.
2	Month	The year and month specified like the following: YYYYMM. So 4 digits for year followed by 2 digits for month. So like the following 200905, meaning the year 2009 and month of May
3	count	An integer representing the number tweets of this hash tag for the given year and month
4	Hash Tag Name	The #tag name, e.g. babylove, mydate, etc.

Here is a small example of the Twitter data that we will use to illustrate the subtasks below:

Token type	Month	count	Hash Tag Name
hashtag	200910	2	babylove
hashtag	200911	2	babylove
hashtag	200912	90	babylove
hashtag	200812	100	mycoolwife
hashtag	200901	201	mycoolwife
hashtag	200910	1	Mycoolwife
hashtag	200912	500	mycoolwife
hashtag	200905	23	abc
hashtag	200907	1000	abc

Using the twitter data set downloaded from LMS please perform the following.

a) [Spark] Find the single row that has the highest count and for that row report the month, count and hashtag name. So for the above small example data set the result would be:

Month: 200907, count: 1000, hash tag name: abc

[6 marks]

b) [Do twice, once using Hive and once using Spark] Find the hash tag name that has tweeted the most in the entire data set. Report the total number of tweets for that hash tag name. So for the above small example data set the output would be:

abc 1023

[12 marks total: 6 marks for Hive and 6 marks for Spark]

c) [MapReduce] Given two months x and y, where y > x. Find the hashtag name that has increased the number of tweets the most from month x to month y. Ignore the tweets in the months between x and y, so just compare the number of tweets at month x and at month y. Report the hashtag name, the number of tweets in months x and y. Ignore any hashtag names that had no tweets in either month x or y. You can assume that the combination of hashtag and month is unique. Therefore, the same hashtag and month combination cannot occur more than once. Take x and y as command line arguments as was done in Task D of Lab 3. For the above small example data set:

Input: x = 200910, y = 200912

Output (hashtag, month x count, month y count):
mycoolwife, 1, 500

[12 marks]

Task 3: Creating inverted index from Bag of Words data [20 marks]

In this task you are asked to create an inverted index of words to documents that contain the words. Using this inverted index you can search for all the documents that contain a particular word easily. The data has been stored in a very compact form. There are two files. The first file is called *docword.txt,* which contains the contents of all the documents stored in the following format:

Attribute number	Attribute name	Description
1	Doc id	The id of the document that contains the word
2	Vocab Index	Instead of storing the word itself. We store the index into the vocabulary file. *The index starts from 1.*
3	count	An integer representing the number of times this word occurred in this document.

The second file called *vocab.txt* contains each word in the vocabulary, which is indexed by attribute 2 of the *docword.txt* file.

The data set used for this task can be found inside the Bag of words directory of the assignment_datafiles.zip on LMS.

If you want to test your solution with more data sets you can download other data sets of the same format from the following source (just need to be careful that you delete the first three lines from the *docword.txt* files that you download):

http://archive.ics.uci.edu/ml/datasets/Bag+of+Words

Here is a small example content of the *docword.txt* file.

Doc id	Vocab Index	Count
1	3	1200
1	2	120
1	1	1000
2	3	702
2	5	200
2	2	500
3	1	100
3	3	600
3	4	122
3	5	2000

Here is an example of the *vocab.txt* file

Plane
Car
Motorbike
Truck
Boat

Using the input files *docword.txt and vocab.txt* downloaded from LMS, complete the following subtasks using spark:

a) [spark] Output into a text file called *"task3a.txt"* a list of the total count of each word across all documents. List the words in ascending alphabetical order. So for the above small example input the output would be the following (the format of the text file can be different from below but the data content needs to be the same):

Boat 2200
Car 620
Motorbike 2502
Plane 1100
Truck 122

[6 marks]

b) [spark] Create an inverted index of the words in the *docword.txt* file and store the entire inverted index in binary format under the name *InvertedIndex*. Also store the output in text file format under the name *task3b*. The inverted index contains one line per word. Each line stores the word followed by a list of (Doc id, counts) pairs (one pair per document). So the output format is the following:

word, (Doc id, count), (Doc id, count), …

i. Note you need to have the list of (Doc id, count) in decreasing order by count.
ii. Note you need to have the words in ascending alphabetical order

So for the above example input the output text file(s) would contain (the actual format can a bit different but it should contain the same content):

Boat (3, 2000), (2, 200)
Car (2, 500), (1, 120)
Motorbike (1, 1200), (2, 702), (3, 600)
Plane (1, 1000), (3, 100)
Truck (3, 122)

For example following format for the text file would also be acceptable:

(Boat,ArrayBuffer((3,2000), (2,200)))
(Car,ArrayBuffer((2,500), (1,120)))

(Motorbike,ArrayBuffer((1,1200), (2,702), (3,600)))
(Plane,ArrayBuffer((1,1000), (3,100)))
(Truck,ArrayBuffer((3,122)))

[9 marks]

c) [spark] Load the previously created inverted index stored in binary format from subtask b) and cache it in RAM. Search for a particular word in the inverted index and return the list of (Doc id, count) pairs for that word. The word can be hard coded inside the spark script. In the execution test we will modify that word to search for some other word. For example if we are searching for "Car" the output for the example dataset would be:

Car (2, 500), (1, 120)

[5 marks]

Measures to minimize shuffle and HDFS costs [15 marks]:

1. The use of combiners for map reduce programs [10 marks]
2. Taking advantage of the fact that data is sorted according to intermediate key to reduce the work of the mapper and/or reducer. [5 marks]

Bonus Marks:

1. Using spark perform the following task using the data set of task 2.

   [spark] Find the hash tag name that has increased the number of tweets the most from among any two consecutive months of any hash tag name. Consecutive month means for example, 200801 to 200802, or 200902 to 200903, etc. Report the hash tag name, and the 1st month count and the 2nd month counts. So for the small example data set of task 2 the output would be:

   Hash tag name: mycoolwife
   count of month 200812: 100
   count of month 200901: 201

   [10 marks]