This exam is **open-book**. Study materials and your own notes are allowed. Pocket calculators are allowed. No other electronics allowed except for the Chromebook!

Write your **name and student number** on the first page (below this box), and in the header of each sheet of paper.

Give your answers **on this paper** in the space provided. Design your answer on **scratch paper before** you start writing here, or you may run out of space! The explanation must be **correct and complete** in order to get all the points for that question. Partial points are possible. The points add up to 90 % (10 % is given for free).

Name and student number: _____

Question 1 *(11 %)*

Say you have a computing cluster with a Distributed File System (DFS). The cluster consists of:

- N **data servers** and one master server,
- each server with D **disk space** and M **memory** (RAM).

The DFS is configured with:

- a **chunk size** C,
- and needs to store T **metadata per chunk**.

If you were to store big data on this cluster, in one big file, what is the **maximum file size** that can be stored on this cluster?

Your answer would be a function of the variables above, for example written like: $(N \cdot D)/C$. Explain your answer. If you need to use other variables besides those listed above, just explain what they are.

Answer:

Question 2 *(14 %)*

This question is about understanding and comparing **local** with **distributed** execution in Spark. Assume you have a computing cluster, some big data stored on it, and your Spark program which reads *input data* from disk, processes it, and writes *output data* back to disk.

(a) Say you need to execute your Spark program in **local** mode. Answer the following questions briefly:

- What Spark command(s) could you use to start such an execution?
- Where in the cluster will the code execute?
- Where in the cluster will the *input data* be fetched from?
- Where in the cluster will the *output data* be stored?

(b) Say you need to execute your Spark program in **distributed** mode. Answer the same questions from (a).

(c) What are the advantages of executing the program in **local** mode (if any)?

(d) What are the advantages of executing the program in **distributed** mode (if any)?

Question 3 *(12 %)*

Answer the following questions about the design of Spark:

(a) What design choice(s) might make a Spark program **faster to execute** than some MapReduce code which does the same calculations?

(b) Say that you have some input data in a simple `.csv` format. What design choice(s) might make a Spark SQL program **faster to execute** than a core Spark program which does the same calculations on the same input data?

(c) In Spark, the execution is lazy. Which of these Spark operations would **trigger an execution** of the program? You can simply underline them here, then explain your choice briefly in the space below:
`df.write.save()`, `df.groupby()`, `df.show()`, `df.coalesce()`, `df.drop()`, `df.select()`.

Question 4 *(13 %)*

This is about storing big data reliably on a cluster. Answer the following questions:

(a) What storage frameworks that you know are able to **store big datasets reliably and long-term** on a cluster? Simply name them.

(b) What types of **write operations** are supported in each of these storage frameworks? In other words, where in the dataset can each of the frameworks write data?

(c) Here is a case study. A company wants to store a large number of **user emails**. When users write new emails, the company needs to add them to the dataset. When users delete any of their old emails, the company needs to erase that data from the dataset. **Which storage framework** would be the best choice? Explain why, and (briefly) how you would organise the data in this type of storage. You can assume an email has: a sender, one or more destinations, a body of text, and a date.

Question 5 *(8 %)*

This is about the Bigtable database.

(a) What type of NoSQL data store is Bigtable?

(b) What design features make Bigtable a **fast** database to use for big data? List them briefly below.

(c) What design features (if any) make Bigtable **reliable**? List any below.
(You won't need all the space below.)

Question 6 *(15 %)*

This is about publishing and processing big streaming data with Kafka and Spark Streaming.

(a) In Kafka, for what practical advantage(s) are topics divided into **partitions**?

(b) In Kafka, roughly how many consumer applications could be accommodated at the same time per topic?

(c) What is the difference, if any, between the notion of **partition** in Spark Streaming, and the same in Spark?

(d) What is the difference, if any, between the set of **transformations** available in Spark Streaming, and those available in Spark?

Question 7 *(17 %)*

This asks to write (from scratch) basic code for training a distributed Decision Tree *binary classifier* in Spark. You have a training dataset D. It consists of N records (or rows, or samples), with N big. Each record has $n+1$ numbers: n *features* which are real numbers (the picture has $n = 2$), and one *label* (of value 1 or 2). Say the dataset was already loaded from disk into Spark, using a library of your choice (specify which).

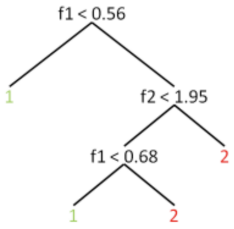
1. Gini Index Write a Spark function `GiniIndex()` which takes in input D and outputs a real number as follows. For any class c, f_c is the fraction of records from that class in D. The Gini Index is $G = \sum_c f_c \cdot (1 - f_c)$.

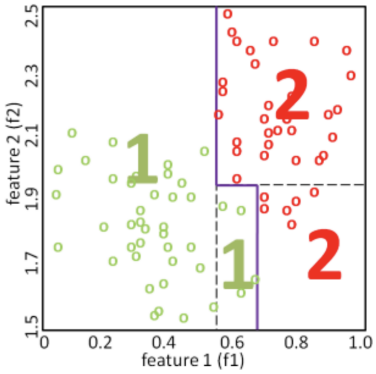
2. Gini Index for a split A split is an internal node, defined by a feature f_i (say, f_1) and a threshold t_i (say, 0.56). There are three splits in the picture. Write a Spark function `Split()` which takes D, f_i, and t_i, and outputs a real number as follows. Split D into two parts: (a) all records where $f_i < t_i$, and (b) the rest. G_a is the Gini Index for part (a), and w_a is the fraction of records from D which is in part (a); the same for (b). The Gini Index for this split is then $G_a \cdot w_a + G_b \cdot w_b$.

3. Training a tree This is a greedy procedure: Start with D. Then propose (up to you how) a number of splits to be tested, across all features. The split with the *lowest* Gini Index is retained. It becomes a new node in the tree. Iterate on both splits of this node. The iteration stops if the Gini Index for a proposed split is higher than the Gini Index of the unsplit (or "parent") dataset. Sketch code for this greedy procedure (as much as you can).

State whether your code would run **efficiently** in distributed mode. Does it have any limitations?

Answer:





Answer (continued):