This exam is **closed-book** (no electronics allowed, including pocket calculators; no paper materials allowed, except for those handed by the teacher).

Write your **name and student number** on the first page (below this box). In case any of the pages get detached from the rest, also write your student number in a header of each sheet of paper.

Each question is marked with a number of percentage points (for example, 10 %), and they add up to 100 %. 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 on this paper. Handwrite neatly.

You receive the points based on the *correctness and completeness* of each answer. No question is mandatory (any could be left blank). There is no penalty for incorrect answers, and partial points may be awarded for an incomplete answer.

Name and student number: ____________________

Question 1 *(15 %)*

Encircle the **T** (True) or **F** (False), and, to get the full points, explain each answer briefly.

In the **Google File System**:

1. **T** **F** A client application can append to existing files.
   Why:

2. **T** **F** Client applications need to know the chunk index in order to read or write in a file.
   Why:

3. **T** **F** The master keeps a record of all chunk locations at all times.
   Why:

Question 2 *(14 %)*

The frameworks for the *distributed storage* of big data (that you know from this course) have a design that is at least partly centralized (in other words, have a single managing machine). Answer concisely the following questions:

1. List the names of the frameworks for distributed storage of big data that you know.
   Answer:

2. For each such framework, list what aspects of the system (storage of what? which operations?) are indeed centralized.
   Answer:

3. For each such framework, what (if any) measures are taken when the "master" machine fails, to achieve fault tolerance?
   Answer:

Question 3 *(18 %)*

Here's a short quiz on practical matters related to the **Spark** big-data processing framework:

1. Is an *RDD partition* big data?
   Answer:

2. List two ways in which the Spark `map` operation is different than the MapReduce `map` task.
   Answer:

3. What is a *task* in Spark vocabulary?
   Answer:

4. For a hypothetical Spark job executed on a cluster with the options `--master yarn --deploy-mode cluster`, estimate *the number of computing cores* that this job will occupy, based on your theoretical knowledge or practical experience. (You can state this is any terms you see fit: you may refer to the size of the input data, to the number of worker machines in the cluster and the size of their resources, etc.)
   Answer:

5. Why is a Spark `join` operation expensive?
   Answer:

6. If a Spark `join` operation is really needed, what can the programmer do to make this operation less expensive?
   Answer:

Question 4 *(10 %)*

Say you are hired by a researcher in linguistics to solve the following data problem: given a large dataset of all Wikipedia webpages in English (in plain text), compute *sets* of English words from this encyclopedia which are *anagrams* of each other (or: consist of the same, but rearranged, letters). An example of such a set is {`soup, opus, oups`}.

Sketch a **MapReduce** program (in pseudocode, Python, or simple English) which takes in input the Wikipedia dataset, and outputs all the sets of anagrams in this data. Each webpage is stored in a file, and each `Map` task reads one such webpage as a tuple `(URL, text)`.

Question 5 *(16 %)*

Answer briefly the following questions on the distributed stream processing frameworks **Storm** and **Spark Streaming**:

(a) What is the important difference between the basic *data types* processed on cluster machines in these frameworks?
Answer:

(b) How is *intermediate data stored* in these two frameworks?
Answer:

(c) How do these processing frameworks differ in terms of the *processing speed* (throughput and latency), and why is that?
Answer:

(d) How do these processing frameworks differ in terms of the reliability *guarantees* they provide? Can you say that one of these frameworks has better reliability than the other?
Answer:

Question 6 *(20 %)*

Design a **Bigtable** schema for a database suitable to use at Twitter. The database should store Twitter's 330 million monthly *users* and their various metadata, their *tweets*, and the identity of users who are *following* / are *followed by* other users. (You can ignore here other Twitter data, such as retweets.)

The Bigtable schema will be managed with an API; here are some operations from this API:

1. `put_tweets(userid, tweets)` adds a user's new tweet(s) to the database;
2. `put_followers(userid, followers)` adds a user's new followers to the database;
3. `get_recent_tweets(userid)` returns a user's most recent 1000 tweets;
4. `get_followers(userid)` returns a user's followers;
5. `get_popular_hashtags(userid)` returns the user's most-tweeted hashtags;
6. `get_topic(hashtag)` returns a random selection of 1000 tweets which contain that hashtag.

New tweets are incoming at great speed, so the design should have a *fast* API; this is your main performance factor.

- Give a textual and/or visual description of the **Bigtable schema**, consisting of the table(s) needed, the column families, and examples of rows and columns (with values).
- List concisely the **Bigtable operations** needed to execute these Twitter API operations.

Answer:

(Answer continued)

Question 7 *(7 %)*

Please give the teacher one good suggestion on what to change in order to improve this course next year.

Some changes will definitely be made: (a) the *project* will start one week earlier (so that more work is done in December, rather than be left for January), (b) the project and the exam will have the same *weight* in the final grade (40% each, instead of 30% and 50%), and (c) the Spark version on the clusters should be upgraded.

Answer: