Projects descriptions and indications

You are asked to form groups of 3 students max.

Each group has to chose one of the following problems, design algorithms for the solution, and write a report including:

1. a description of the adopted solution (4 points)
2. comments/description of the designed algorithms, possibly by using examples (4 points); comments to main fragments of code (3 points)
3. experimental analysis, concerning in particular scalability (except for the last project on recommendation systems for which scalability analysis is not required) (3 points)
4. comments about the experimental analysis outlining weak and strong points of the algorithms. (3 points)
5. an appendix including the code. (3 points)

A pdf version of the report has to be sent via email (dario.colazzo@dauphine.fr) before September 15, 2018.

For any question regarding the projects feel free to ask by email if needed. If needed we can arrange for a telephone call or a meeting in July (I am absent from June 30 to July 8).

IMPORTANT, for all projects: you are allowed to consult available documentation on the Web in order to learn required algorithms, in particular for the fourth project.
A suggested text is *Data-Intensive Text Processing with MapReduce,* by Jimmy Lin and Chris Dyer, freely available on the web, Just google the title plus 'pdf'.

Matrix multiplication. Consider the problem of multiplying two big matrixes $A_{n,m}$ and $B_{m,n}$. Design MapReduce algorithms in Spark for this problem, knowing that up to two jobs may be needed. Consider the simple textual representation of a matrix where the value v for $A_{i,j}$ is represented by a text line 'i j v'. Only non-zero values will have a line in the textual file representing a matrix. Provide at least two non-trivial different algorithms, and implement them in Spark. Perform experimental evaluation by considering 5 couples of matrixes of increasing sizes, by doubling the sizes from a pair of matrix to the subsequent pair. You can use Spark to generate big matrixes. In case of problems with your computer resources, the size of matrixes can stay under 1 GB.

TF-IDF. Consider the problem of calculating the TF-IDF score for each word in a set of documents for each document. Many sources on the web describes the sequential TF-IDF algorithm.
Provide a MapReduce algorithm in Spark to calculate TF-IDF scores given an input set of documents. Provide both a MapReduce Python Hadoop streaming and a Spark implementation.
Perform experimental analysis in order to compare performances of the two implementation. To test scalability consider 5 document sets of increasing sizes. For instance, size can double from a set to another.
The set of input textual document can be either dowloaded from the Web or generated by a Python script for instance. To speed up document generation Spark can be used. Words of documents can be randomly picked from an input fixed vocabulary.

Words co-occurrence matrix. Consider the the problem of building a word co-occurrence matrice from large corpora of textual documents. Formally, the co-occurrence matrix is a square n X n matrix where n is the number of unique words in all the documents (i.e., the vocabulary size). A cell m_{ij} contains the number of times word w_i co-occurs with word w_j within a specific context—a natural unit such as a sentence, paragraph, or a document; for simplicity consider that the context is a document, and consider an input collection of relatively small documents, that you can generate starting from a fixed vocabulary. MapReduce/Spark can be used to generate the

documents. Note that the upper and lower triangles of the matrix are identical since co-occurrence is a symmetric relation.
Implement a Spark algorithm for this problem in Python. Performs scalability analysis on five document sets of increasing sizes. For instance, size can double from a set to another.
Also, you can use Spark to make document generation easier.

Single-source shortest path problem. The task is to find shortest paths from a source node to all other nodes in a directed graph. This problem is solved by the Dijkstra's algorithm, which is sequential.
The project has a double purpose. First get familiar with Dijkstra's algorithm, then devise a MapReduce version of the algorithm in Spark. As you will realise, the process is actually iterative, so the Spark algorithm must be iterative too.
Provide a Spark implementation of the algorithm, and test it on a simple graph you will provide.
Then perform scalability experiments on graphs of increasing sizes as for previous projects. You can use Spark to generate the graphs.

Warning: this project is particularly difficult as it requires the study and understanding of scientific paper below indicated.
Simple recommender system based on latent classes - Probabilistic Semantic Latent Indexing (PLSI)
Let us consider a dyadic dataset, composed of N users U = {u1, … uN} and M items I = {i1, …, iM}. For each (user, item) pair, you know if a user interacted with or liked an item. In this project, you will implement a basic recommendation algorithm based on Hoffman (1999) work, to recommend items to users.

Hoffman (1999) describes a simple probabilistic algorithm based on latent classes. Latent classes are unobserved classes, clustering both users and items. The authors assume a multinomial distribution of users given latent classes, and of items given classes. Users and item can then belong to several latent classes. Latent classes are expected to gather similar items and users, for example, there can be a class gathering sci-fi fans and Star Trek movies.
Parameter estimation is done using the EM algorithm, which is a standard procedure when working with latent parameters.

This algorithm was one of the building bricks of Google News recommender system. Das (2007) describe Google's MapReduce implementation of the basic PLSI algorithm (see section 4.2).

You are asked to implement in *Spark* the PLS algorithm, based on Das (2007) MapReduce formulation, and to apply it on the well known Movielens dataset.
Note that the Movielens dataset contains ratings ranging from 1 to 5. You can reduce this information to seen/not seen in order to use the basic version of PLSI. If you are very motivated, you can try to integrate preference values to PLSI, as described in Hoffman (1999). You can also compare your results to Spark LDA implementation and discuss the use of LDA versus PLSI.

---

[1] Hofmann, Thomas, and Jan Puzicha. "Latent class models for collaborative filtering." IJCAI. Vol. 99. No. 1999. 1999. – http://people.eecs.berkeley.edu/~russell/classes/cs294/f05/papers/hofmann+puzicha-1999.pdf
[2] Das, A. S., Datar, M., Garg, A., & Rajaram, S. (2007, May). Google news personalization: scalable online collaborative filtering. In Proceedings of the 16th international conference on World Wide Web (pp. 271-280). ACM. –
https://pdfs.semanticscholar.org/fb42/07376177f18a6cf58b53ecd231fb3395ca33.pdf
[3] https://grouplens.org/datasets/movielens/
[4] https://spark.apache.org/docs/2.2.0/api/python/pyspark.ml.html#pyspark.ml.clustering.LDA