Heidelberg University
Winter semester 2017/2018

Group Parallel and Distributed Systems (PVS)
Artur Andrzejak, Diego Costa

Problem Set 3 for lecture Mining Massive Datasets

Due November 13, 2017, 11:59 pm

Exercise 1 (7 points)

Consider the incomplete implementation of an algorithm for hierarchical clustering in Spark shown in lecture 3.

1. Complete the implementation and improve the code for routine "merge" by avoiding the loop in computation of pairwise distances ("Step 1 of 4"). Hint: investigate Spark documentation[1] for suitable transformations. Submit your modified source code as the solution **(5 points)**.
2. Extend your implementation such that it loads data from the file (provided in Moodle) `dataset-problemset3-ex1-2` . Run your code and print the sequences of cluster merges as a result. Submit this sequence as your solution **(1 point)**.
3. Run your code on datasets containing random 2-dimensional points generated by routine `generateData(N, k=5)`[2] for various values of N. What is the (approximate) smallest N (named N^*) such that your code needs at least 5 minutes? Describe how you find out the value of N^* and your system performance indicators (CPU, memory, ...) during the investigation. Submit your source code and the timing results (i.e. execution time for all N's you have tried) as your solution **(1 point)**.

Exercise 2 (6 points)

Create a pseudocode implementation for a hypothetical Map Reduce framework to solve the following problems.

1. **k-means Clustering**
   You can assume that our Map Reduce framework offers a while-loop that executes the Map and Reduce phases as long as some condition holds. See lecture 2 for details on k-means.
2. **Inverted Indexing**
   Given a set of documents, an inverted index is a dictionary where each word is associated with a list of document identifiers pointing to documents in which that word appears. Say, you have a huge database of documents and there is a list of popular keywords that you want to create an inverted index for. Each Mapper should get a part of the database as two-element lists $[doc_i, text_i]$ and a copy of the list of keywords. The output of the Reducer(s) should be of the form $(keyword_i, [doc_i^0, doc_i^1, \ldots])$.
   Does your solution scale well with the number of keywords?
3. **(Inner) Join**
   Recall the Map Reduce implementation of an (inner) Join from lecture 4 (slide

[1] Spark Documentation (Transformations)
[2] Function is available in the kmeans code provided on IMMD Public Code or on Moodle at K-Means Source Code - Lecture 2

"Map-Reduce Join"). This implementation creates a lot of network traffic, because of the shuffling. When two databases have only a very small inner join, still almost all the data has to be transferred over the network. Create a new implementation with the assumption that one of the two databases fits easily into the memory of each node. Avoid causing network traffic, especially when the two databases have only a very small inner join.

Exercise 3 (3 points)

Frameworks for distributed data processing like Google MapReduce, Apache Hadoop or Apache Spark borrow terms from functional programming. One can use functions called map, reduce or filter. However, there are semantic differences in definition, depending on the framework.

a) Explain the differences between map in Map-Reduce and Spark, in terms of the input/output. Explain also the differences for reduce.

b) Write a piece of code in Spark which emulates the Map-Reduce flow (one phase).

c) Emulate Spark's reduce as one Map-Reduce phase.

Exercise 4 (Bonus, 4 points)

Read Sections 2.5, 2.3.2 and 2.3.6 in the book *Mining of Massive Data Sets*[3]. Then solve exercises 2.5.1 a) and b): What is the communication cost of each of the following algorithms, as a function of the size of the input relations, matrices, or vectors?

a) The matrix-vector multiplication algorithm of Section 2.3.2.

b) The union algorithm of Section 2.3.6.

Exercise 5 (Bonus, 3 points)

Describe the key ideas of the paper *Pairwise Element Computation with MapReduce*[4]. The pdf can be downloaded if your browser is in the university domain (physically or via VPN).

[3] Available for free online: http://www.mmds.org/#ver21

[4] Pairwise Element Computation with MapReduce