Computing at Scale in Machine Learning: Distributed computing and algorithmic approaches (14038)

B-TU Cottbus-Senftenberg · WiSe 2023
Prof. Dr. Alexander Schliep, Nathalie Gocht, M.Sc. and Aleksandra Khatova, M.Sc.

Problem set 2 from November 2, 2023 · Due on November 10, 2023

Problem 1 (Parallel k-means, 6 Points)**.** Prepare the parallel implementation of the k-means algorithm using the `multiprocessing` Python package. Use the k-means implementation in `kmeans.py` as the starting point.

a) (2 pt) Describe the building blocks of the k-means algorithm, which building blocks possibly can be parallelized, where and what data has to be exchanged, and whether sequential bottlenecks (in the sense that they cannot be parallelized) possibly exist. The description can be provided as a brief sketch of the method in pseudo-code, or by annotating (e.g. drawing over) a listing of kmeans.py .

b) (2 pt) Measure the overall running time and the individual running times of the sections of the serial program you intend to parallelize. Use at least 10,000 sampled points for k-means.

c) (2 pt) For the two sections which contribute the most to the total running time: Use Amdahl's law to compute the theoretical total speedup assuming 4, respectively 8, cores.

Problem 2 (Extension Monte Carlo, 4 Points)**.** Extend your solution to Assignment 1, Problem 2.c to use an accuracy goal to determine when to stop the simulation. That is, instead of the `--steps` command line argument your code should have an `--accuracy` command line argument, and it should run the simulation in parallel until this accuracy goal for approximating π is achieved. This requires the main thread to generate new jobs for the workers based on the outcome of previous jobs. Do not use the `p.map` command repeatedly. Instead use multiprocessing's Queue or JoinableQueue classes for communicating the input and output to the workers. Please submit your code as a file named `problem2_2.py`. Hint: convergence speed depends on random number seeds in each thread and thus it makes little sense to compute speedup by just measuring the wall clock time until convergence. Instead, use samples per second of real-time as the metric to measure the speedup. Produce a plot similar to Problem 2.b in Assignment 1.

Note: Please hand in:

- A PDF containing:
  - the answers to problem 1)
  - the plots to problem and possible explanations 2)
- A Python script named `problem2_2.py`

Python code must be handed in as '.py'. Code submission in PDF will be graded with 0 points. Your code files can be accompanied with PDF.

You are allowed to use the example solutions provided where they might be helpful for answering question on this problem set.

Please feel free to discuss the problems and approaches to solving them with other students. However *you* should understand results of discussions well enough to write up the solution by yourself and possibly explain the solution. When you do collaborate, please make other's contributions clear. Present your solutions concisely.