Techniques for large-scale data (DIT871/DAT345)

CSE, Gothenburg University | Chalmers, Period 4, 2018
Graham Kemp, Alexander Schliep (alexander.schliep@cse.gu.se)

Problem set 1 from April 11, 2018 · Due on April 18, 2018

Project 1 (Monte Carlo, 4pt)**.** Modify `mp-pi-montecarlo-pool.py` s.t. you can specify an accuracy goal of the simulation; e.g., instead of the `--steps` parameter specifying a fixed number of steps, implement an `--accuracy` parameter. The simulation should still use the number of workers specified by the `--workers` argument, but generate parallel jobs until the accuracy w.r.t. to 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 `Queue` or `JoinableQueue` for communicating the input and output to the workers.

Project 2 (k-means, 6pt)**.** Implement a parallel version of the k-means algorithms using the `multiprocessing` Python package . You can use the implementation on Canvas `kmeans.py` as the starting point. Your file has to contain a function `kmeans(k, data, nr_iter)` which takes an integer number of clusters `k`, a numpy matrix `data` and an integer number of iterations `nr_iter` as arguments and returns the total variation of the clustering and a cluster assignment. Please use a `if __name__ == "__main__":` conditional before any statements in the Python file, so that your code can be imported without starting computations.

The goal is to maximize speedup by parallelizing k-means to the largest extend possible. Note, we will run your code.

1. Describe the structure of the algorithm and which segments can be run in parallel. Clarify when does data need to be exchanged etc. The description could be provided as a brief sketch of the method in pseudo-code.

2. Measure the overall running time and the running times of the parts of the serial program you intend to parallelize. Use sufficiently many data points so that at least 30s is spent within the sections you plan to parallelize.

3. Use Amdahl's law to compute the speedup necessary in the parallel sections to achieve a total speedup of 2 respectively 4.

4. Implement your parallel version of k-means.

5. Measure running times and report the speedup; i.e., the ratio of original and improved running times when using 2, 3, or more cores. Use a large enough number of data points so that the serial version needs at least 2 min running time on your machine. Use the `sklearn.datasets.make_blobs()` to generate data.

6. Repeat the measurement but increase the dimensionality of the data (i.e., the number of features; see `http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html`) from the default 2. Does this effect the speedup depending on the number of cores?