Parallel Machine Learning and Artificial Intelligence

Dr. Handan Liu

h.liu@northeastern.edu

Northeastern University

Northeastern University
College of Engineering

Content

- XGBoost Parallelism
- Hands-on:

- Parallel Machine Learning with Dask
- dask distributed
- Dask ML

XGBoost: a High-Performance ML

- XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework.
- XGBoost provides a parallel tree boosting that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Kubernetes, Hadoop, SGE, MPI) and can solve problems beyond billions of examples.
- A large number of Kaggle players chose it for data mining competitions and won the championship in Kaggle competitions.

XBGoost Website: https://xgboost.readthedocs.io/en/latest/

XGBoost Parallelism

- XGBoost is implemented in C++ to explicitly make use of the OpenMP API for multiple threads parallel.
- The speed of XGBoost is both in:
  - individual trees
  - the input data
- XGBoost algorithm, hyperparameters and cross-validation for achieving high performance.

XGBoost Scikit-Learn API

- Two Scikit-Learn wrapper interfaces for XGBoost:
  - XGBClassifier and XGBRegressor
- The two wrapper classes provide the parameter n_jobs to specify the number of threads that XGBoost can use during training.
- Impact of the number of parallel threads to run XGBoost:
  - Setting n_jobs in XGBoost
  - Using GridSearchCV to tune hyperparameters of XGBoost

https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn

Install XGBoost on Cluster

- On the cluster, install on your $HOME
  - $ source activate py2020
  - (py37) $ conda install -c conda-forge xgboost

- On your local machine, install on Windows, macOS or Linux:
  - conda install -c conda-forge xgboost
  - Or: git clone --recursive https://github.com/dmlc/xgboost
  - Or: (sudo) pip install xgboost

https://anaconda.org/conda-forge/xgboost

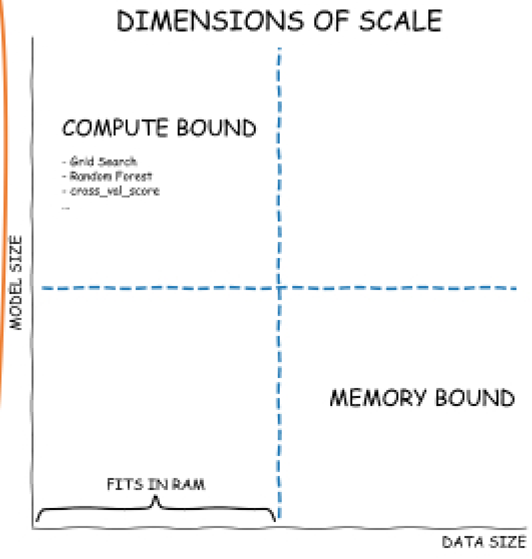
Hands-on: Parallel XGBoost

- Problem description:
  - Otto Group Product Classification Challenge
  - https://www.kaggle.com/c/otto-group-product-classification-challenge
  - This dataset describes the 93 obfuscated details of more than 61,000 products grouped into 10 product categories (e.g. fashion, electrons, etc.). Input attributes are counts of different events of some kind.
  - The goal is to make predictions for new products as an array of probabilities for each of the 10 categories and models are evaluated using multiclass logarithmic loss (also called cross entropy).
- Parallel XGBoost with Joblib and Multiprocessing

Scaling Challenges in Machine Learning

Solution:

- You'd continue to use the collections you know and love (like the NumPy ndarray, pandas DataFrame, or XGBoost DMatrix) and use a Dask Cluster to parallelize the workload on multiple nodes.
- The parallelization can occur through the integrations like Dask's joblib backend to parallelize Scikit-Learn directly or Dask-ML's estimators



Solution:

- You'd use one of Dask's high-level collections like (Dask Array, Dask DataFrame, or Dask Bag) combined with one of Dask-ML's estimators that are designed to work with Dask collections.

Parallel Machine Learning with Dask

- Dask Scheduling
  - Single machine scheduler:
    - ✔ threaded
      - Default, no setup, no costs. Parallelism for the computation which is dominated by non-Python code, NumPy arrays, Pandas DataFrames, or any of the other C/C++/Cython based projects, like XGBoost, etc.
    - ✔ multiprocessing
      - No setup. Bypass GIL. Introduce performance penalties.
      - Idea for the workflows are relatively linear (small inter-task data transfers) as well as small inputs and outputs
    - ✔ synchronous
      - Execute all computations in the local thread with no parallelism at all. Good for debugging and profiling,
  - Distributed scheduler
    - The Dask distributed scheduler can either be setup on a cluster or run locally on a personal machine.

Single Machine: dask.distributed

- The Dask distributed scheduler can either be setup on a cluster or run locally on a personal machine. Despite having the name “distributed”, it is often pragmatic on local machines for a few reasons:
  - It provides access to asynchronous API, notably Futures
  - It provides a diagnostic dashboard that can provide valuable insight on performance and progress
  - It handles data locality with more sophistication, and so can be more efficient than the multiprocessing scheduler on workloads that require multiple processes.

dask.distributed

- Difference between threads vs. processes in dask.distributed
- What are the meaning of “processes=False”, “n_workers”, “threads_per_worker”?

Refer to: https://docs.dask.org/en/latest/setup/single-distributed.html

•Stay safe!
•See you next class!

<u>Next Lecture will Continue:</u>

Hands-on Lab

Project Process