15618: Spring 2023 - Final Project Proposal

Wei-Lun Chiu (weilunc) (weilunc@andrew.cmu.edu)
Jhao-Ting Chen (jhaoting) (jhaoting@andrew.cmu.edu)

March 28, 2023

1 Title

Supercomputer or Cloud Computing: Large-scale, High-computation Data Clustering Tasks with PSC/GHC+PyOMP and AWS+PySpark

2 URL

https://jtchen0528.github.io/cmu-15618-project/

3 Summary

We aim to analyze the performances and costs of executing k-means data clustering, between executing on a supercomputer (PSC/GHC with PyOMP) and on a set of workers with comparable resource constraints (AWS instances with PySpark). We aim to analyze the serial and parallelized implementation, on performance, data imbalances, different algorithms.

4 Background

4.1 Data Clustering - K-means

The dataset size for machine learning tasks grow larger by days, and self-supervised tasks become more popular than before. The performance and effectiveness for data clustering tasks such as feature reduction or pseudo-labeling hence raise more research awareness. reference in research. For our project, we choose to analyze a common clustering algorithm, k-means clustering, and its variations (grid-based, centroid-based, partition-based and density-based). We will implement serially and in parallel, on multi-core machine and multi-node cluster.

4.2 Cloud Computing

Cloud computing provides a cost-efficient alternative to maintaining expensive high-performance computing infrastructure. With on-demand computing resources, scalability, and pay-as-you-go services, one may access computing power as needed without the need for investing in specialized hardware. Cloud computing is especially beneficial for those who don't have access to resources like PSC or multi-core machines at home. By executing large-scale, high-computing tasks on several cloud service instances in parallel with distributed technologies, organizations can achieve comparable performance and accurate results while also significantly reducing costs.

4.3 PyOMP

PyOMP [1] caught our eyes when we're researching through our project. PyOMP, released by *Intel Corp.*, provides an easy-to-use API that allows developers to parallelize their Python code with minimal modifications, by adding OpenMP directives to their code. Instead of embedded parallelism inside Numpy [2], which has overhead when creating and destroying threads, PyOMP library was implemented in Numba. Numba utilized JIT compiler to generate optimized machine code at runtime, provides significant speedups for computationally intensive tasks. PyOMP then offers users an OpenMP-similar syntax for accessibility. Programs written in C with OpenMP performs only 2.8% faster than programs in PyOMP.

4.4 PySpark

PySpark [3] is a Python API for Apache Spark [4], a powerful open-source distributed computing system used for big data processing and analytics. Spark is designed to work with large-scale data processing tasks that require high-speed data processing and distributed computing capabilities. PySpark provides even more easy-to-use interface for users to handle data analytic tasks in Python. For our project, we aim to implement our program in PySpark, so that data can be clustered in parallel on multiple cloud instances.

5 The Challenge

The challenges can be described in the following bullet-points:

(a) **New parallelization libraries** Familiarize with PyOMP and PySpark, implement parallelized algorithm for k-means clustering algorithm.

(b) **Several parallelization algorithms** Implement different parallelized methods of the k-means clusting algorithm, such as grid-based, centroid-based, partition-based and density-based. Inspect workload imbalances for each processor or cloud instance. With PyOMP and PySpark.

(c) **Explore communications in the program** The data were divided for each processors/instances. The communication per computation for the same program may differ when the number of workers scale up. Experiments are required for proper explanation.

(d) **Cloud service set-up** Configure instances on cloud services with similar hardware that could be compared with PSC or GHC. Configure PySpark environments on master instance and slaves.

(e) **Scalability on processors** Implement and analyze the scalability of the processors executed, within or across machines.

(f) **Constraints in supercomputer and cloud instances** Discover execution constraints in single high-compute machine or several cloud instances, in memory, disk, network bandwidth, etc.

6 Resources

7 Goals and Deliverables

Our step-by-step goals are listed below:

(a) Implement k-means serial algorithm in python, and parallelized version with PyOMP. Sample datasets would be MNIST [5], CIFAR-10 [6], and CIFAR-100.

(b) Conduct experiments on GHC and PSC with different number of processors, analyze the speedups. Expecting speedups to be in proportion to the number of processors executed on.

(c) Implement different k-means strategies such as grid-based, centroid-based, partition-based, or density-based for performance comparison and workload imbalances inspection.

(d) (Optional) Implement serial and parallelized program in C. Analyze the performance between OpenMP and PyOMP.

(e) Implement k-means distributed/parallelized algorithm with PySpark, on set of instances with similar hardware constraints added up as GHC/PSC.

(f) Implement different k-means strategies such as grid-based, centroid-based, partition-based, or density-based for performance comparison with PySpark.

(g) Conduct experiments on performance difference, data communication overheads, and results between PySpark and PyOMP, with different number of workers.

(h) (Optional) Scale up the problem with larger datasets. Repeat the experiments.

(i) Conclude the performance, costs, complexity trade-offs between renting/maintaining a super-computer/cloud services.

For our final presentation, we will demonstrate the clustered results, the performance comparison, workloads across processors/instances, costs, ... etc for every implementations we have.

8 Platform Choice

We are running our code in Unix environment. We code in python for consistency when comparing with PySpark and PyOMP, also python is less complex for data analysis tasks. We will run our code with the following processors:

(a) Intel Core i7-9700 CPU on GHC machines.

(b) Two AMD EPYC 7742 CPUs on PSC regular memory machines.

(c) Sets of AWS spot instances with single-node performance similar to a core in the above two CPUs.

9 Schedule

References

[1] Todd Anderson and Tim Mattson, "Multithreaded parallel Python through OpenMP support in Numba," in *Proceedings of the 20th Python in Science Conference*, Meghann Agarwal, Chris Calloway, Dillon Niederhut, and David Shupe, Eds., 2021, pp. 140 – 147.

[2] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sept. 2020.

[3] "Pyspark," https://github.com/apache/spark/tree/master/python, 2017.

[4] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al., "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[5] Li Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[6] Alex Krizhevsky, "Learning multiple layers of features from tiny images," 2009.