Carnegie Mellon University in Qatar

Distributed Systems

15-440 - Fall 2018

Project 3

Out: November 05, 2018

Due: November 26, 2018

Contents

1 Intended Learning Outcomes

This project applies the theory of the popular programming model, *Message Passing*. The main learning outcome of the project is to apply Message Passing Interface (*MPI*), a library standard for writing message passing programs, to a popular real problem, namely cluster analysis using the *k-Means* algorithm.

2 Project Objectives

The overall goal of this project is to get a clear understating on how to apply *MPI* to real problems. We have chosen a clustering analysis algorithm (i.e., *k-Means*) due to its significance and importance in various domains including, but not limited to, data mining and statistical data analysis. For whatever domain our students will be in, the chances are that sooner or later they will run into a clustering problem. The project potentially provides our students with a practical experience augmented with a methodology for solving clustering and other similar problems on a distributed system using *MPI*. The students will also conduct and analyze some scalability studies on various degrees of parallelism and data set sizes.

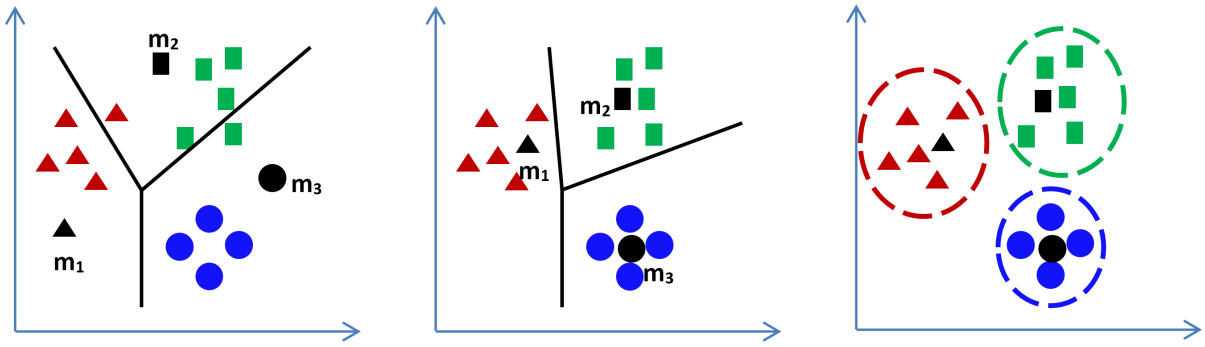
3 Cluster Analysis

Cluster analysis or **clustering** is the task of assigning a set of objects into groups (called **clusters**) so that the degree of similarity can be strong between members of the same cluster and weak between members of different clusters. In short, clustering has to define some notion of "similarity" amongst objects. The objective is to maximize intra-cluster similarity and minimize inter-cluster similarity.

Clustering problems arise in many different applications such as visualization (e.g., visualizing the stock market data to give individuals/institutions useful information about the market behavior for investment decisions), data mining and statistical data analysis including machine learning, pattern recognition, image analysis, information retrieval, and bioinformatics.

Among clustering formulations that are based on minimizing a formal objective function, perhaps the most widely used and studied one is *k-Means* algorithm. Simply put, *k-Means* is an iterative algorithm that attempts to find k similar groups in a given data set via minimizing a mean squared distance function. Initial guesses of k means $(m_1, m_2, , m_K)$ is initially made (see Figure 1(a)). These estimated means are then used to classify the data set objects into k clusters. Afterwards, each mean is recomputed so as to reflect the true mean of its constituent objects (see Figure 1(b)). The algorithm keeps iterating until the recomputed means (almost) stop varying (see Figure 1(c)).

Handout continues on the next page(s)



(a) Initial Means. (b) Recalculated Means. (c) Final Clusters.

Figure 1: k-Means Example.

In this project we will apply *k-Means* clustering to two different applications; **data points** in a 2D plane and **DNA strands** in biology.

Consider a case of a data set composed of data points in ***d-dimensional*** space R^d. In *k-Means* clustering, we specify a set of n data points and an integer k. The problem then is to determine a set of k points in R^d, called *centroids*, so as to minimize the mean squared distance from each data point to its nearest center. In pseudo-code, it is shown by Alpaydin (*Introduction to Machine Learning, page 139*) that *k-Means* essentially follows the following procedure:

```
Repeat:
    For all x^d in X
        b_i^d ⇐ 1 if ||x^d - m_i|| = min_j ||x^d - m_j||
        b_i^d ⇐ 0 otherwise
    For all m_i, i = 1, …, k
        m_i ⇐ (sum over b_i^d x^d)/(sum over b_i^d)
    Until m_i converge
```

Explained in plain English, **k-Means** roughly follows this approach:

1. We start by deciding how many clusters we would like to form from our data. We call this value k. The value of k is generally a small integer, such as 2, 3, 4, or 5, but may be larger.

2. Next, we select k points to be the centroids of k clusters which at present have no members. The list of centroids can be selected by any method (e.g., randomly from the set of data points). It is usually better to pick centroids that are far apart.

3. We then compute the *Euclidean distance* (the similarity function with a data set of data points) from each data point to each centroid. A data point is assigned to a cluster such that its distance to that cluster is the smallest among all other distances.

4. After associating every data point with one of k clusters, each centroid is recalculated so as to reflect the true mean of its constituent data points.

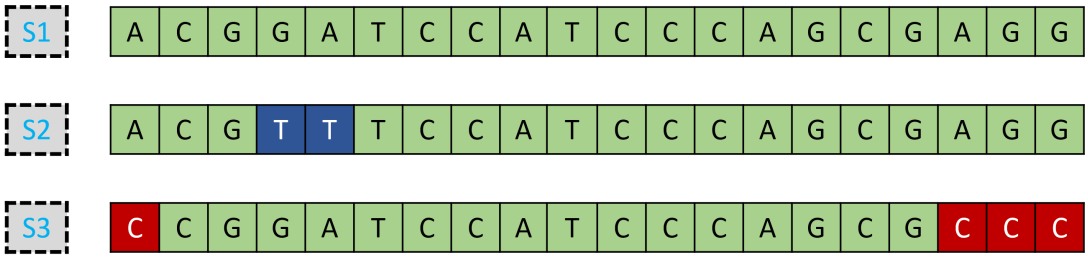
5. Steps 3. and 4. are repeated for a number of times (say, μ); essentially until the centroids start varying very little.

The positive integer μ is known as number of *k-Means* iterations. The precise value of μ can vary depending on the initial starting cluster centroids, even on the same data set.

In this project, you will provide sequential and parallel implementations of the above *k-Means* algorithm with a data set of data points as input and k centroids as output.

4 Clustering DNA Strands

Bioinformatics involves the manipulation, searching, and data mining of biological data, and this includes DNA sequence data. A strand of DNA consists of a string of molecules called bases, where the possible bases are adenine (A), guanine (G), cytosine (C), and thymine (T). We can express a strand of DNA as a string over the finite set **A, C, G, T**. String searching or matching algorithms, which find an occurrence of a sequence of letters inside a larger sequence of letters, or simply match two sequences of letters, is widely used in genetics (e.g., for studying various phylogenetic relationships and protein functions). In many studies, we often want to compare the DNA of two (or more) different organisms. One goal of comparing two strands of DNA is to determine how "similar" the two strands are, as some measure of how closely related the two organisms are. **Similarity in such a scenario can be defined as a function F(. , .) of the number bases in a strand subtracted from the number of changes required to turn one strand into the other.** For example, consider the following three DNA strands:



The similarity between S1 and S2 is denoted as **F(S1, S2)** and is equal to *18*. On the other hand, **F(S1, S3)** = *16*. The *k-Means* algorithm, described in the previous section, can be applied to DNA strands with this given similarity function **F(. , .)** to compare DNA of two or more different organisms.

In this project, you will provide sequential and parallel implementations of the K-Means algorithm with a data set of DNA strands as input and K centroids as output.

Handout continues on the next page(s)

5 Implementation Guidelines

As stated earlier, in this project, you will provide sequential and parallel implementations for **K-Means** with two types of data-sets; a data set of *data points* and a data set of *DNA strands*. For simplicity we assume 2D data points. Furthermore, we assume that strands in the DNA data set are equal in size, and that strands in the list of centroids are also equal in size to each other and equal in size to every strand in the data set.

For the sequential implementation, use C, C++, or Java. For the parallel implementation, you will provide an **MPI-based** version using **MPICH2**, a high performance and widely portable implementation of the Message Passing Interface (*MPI*) standard. *(Note: since you'll have to use C for the parallel implementation, it is recommended to also use C for the sequential implementation (e.g. you may reuse some code)).*

In addition, you have to write your own data set generator that generates a random number $\boldsymbol{P}$ of DNA strands per cluster for k clusters (use any programming language you like), for a given strand length l. We will provide you with a data set generator that generates a random set of 2D data points.

Your sequential and MPI-based *k-Means* implementations should be tested and run on a data set of 2D data points, generated by the given data set generator, and on a data set of DNA strands, generated by your DNA strands data generator.

6 Experimentation and Analysis

Please conduct and provide the following:

- A comparison between your 2 different K-Means implementations in terms of performance and development effort.
- Three scalability studies for your MPI version on:
  - The number of processes with a fixed data set size (use only the data set of the 2D data points for this study). Specifically, use 2, 4, 8, and 12 processes on 4 virtual machines (VMs).
  - The number of VMs with a fixed data set size (again, use only the data set of the 2D data points for this study). Specifically, use 1, 2, 3 and 4 VMs with a fixed number of processes (e.g., 8).
  - The number of data points in your data set of the 2D data points with a fixed number of processes (e.g., 8) and a fixed number of VMs (e.g., 4). Specifically, use 20 million, 30 million, and 40 million data points.
- A discussion on:
  - Your experience in applying *MPI* to the *k-Means* clustering algorithm.
  - Your insights concerning the performance trade-offs of *MPI* and sequential programming with *k-Means*.
  - Your scalability studies (a detailed analysis on collected results should be provided).
  - Your thoughts on the applicability of *k-Means* to MPI.
  - Your recommendations regarding the usage of *MPI* for algorithms similar to *k-Means*.

7 Rubric

DNA Dataset Generator (5 Points)

5pts
- Generator takes at least the number of clusters and number of points per cluster as parameters (**NOTE**: Simply producing random strands will be awarded 1% only)

Sequential Versions (12 Points)

6pts
- K-Means for 2D data points

6pts
- K-Means for DNA strands

MPI Versions (60 Points)

- K-Means using MPI for 2D data points
  - 1pt The code compiles and runs
  - 1pt Initialize MPI and get the size and rank
  - 3pts The master loads data from a file and sends workers their shares
  - 3pts The workers receive data from the master
  - 6pts The workers and the master apply K-means to their portions of data (adopting a good mechanism to compute new centroids)
  - 5pts The workers send their intermediate data (either the summations and the count, or the count and average) to the master
  - 3pts The master receives intermediate data from workers
  - 3pts The master aggregates data and calculates the new means
  - 3pts The master loops over for a new round
  - 1pt After done, cleanly abort the program using the right MPI functions
  - 1pt Compute the runtime correctly

30pts
- Means using MPI for DNA strands (similar to the above)

Write-up (20 Points)

15pts
- Three scalability studies (plots and analysis)

2pts
- Experience in applying MPI to the K-Means algorithm (trade-offs of sequential vs. parallel)

2pts
- Thoughts on the applicability of K-Means to MPI

1pt
- Paper structure, level of writing, and language

Code Style (3 Points)

3pts
- Method Comments, Block comments, Readability, Dead code, Code Design (**NOTE**: Well-document functions that re-compute 2D and DNA centroids)

8 Deliverables

As deliverables, you must submit the following:

1. An archive containing the following:

   i. DNA data-set generator **named** `data_generator(.c)`:

   - Input:
     - Number of clusters c
     - Number of strands per cluster p_k
     - Length of DNA strand l
   - Output:
     - A file with $c \times p_k$ DNA strands, separated by new lines:
       * Sample output with $l = 10$:

         GTGAAAGGTC
         ACGTACCTTG
         CACTAGGATC
         ...

   ii. **Four programs for the *k-Means* implementation** (2 sequential programs for *k-Means* data-points and DNA strands, and 2 parallel programs for *k-Means* data-points and DNA strands):

   (1) `points_seq(.c)`
   (2) `points_par(.c)`
   (3) `dna_seq.c`
   (4) `dna_par.c`

   They must follow the following specifications:

   - Input (via command line arguments):
     - c: Number of clusters c
     - t: Total number of points or strands provided
     - i: Input file containing the data
     - n: Number of iterations *(if not provided, your program should stop as necessary)*
     - l: length of DNA strand *(only applies for the DNA strands K-Means implementations)*
       * Sample program execution:

```
// Runs K-Means on a dataset of 100 points, 3 clusters,
// data from pointsGenerated.txt for 5 iterations

$ mpiexec -f <Machinefile> -n <Number_of_processes>
        ./your_executable -t 100 -c 3
        -i points_generated.txt -n 5
```

- Output: a file with the following information ***for each cluster*** c_i (all separated by new-lines):
  - The final centroid value for c_i
  - The number of points/strands assigned to the cluster c_i
    * Sample output for 2D points (the above, separated by commas on each line):

```
// Runs K-Means on a dataset of 100 points, 3 clusters,
5.212, 9.880, 400
1.511, 3.201, 320
...
```

  - **The name of the output file should be the name of the program (without the extension) followed by "`_results`":**
    * `points_seq_results`
    * `points_par_results`
    * `dna_seq_results`
    * `dna_par_results`

2. An article with a maximum of 5 pages (similar to research articles) that presents your solution, findings, observations and analysis.

9 Useful References

The following is a pair of potentially useful links that can help you with the Project:

- A K-Means interactive demo
- A tutorial on Message Passing Interface (MPI)

10 Submission

Submit your code using AFS (Andrew File System):
`/afs/qatar.cmu.edu/usr10/mhhammou/www/15440-f18/handin/p3/`*`userid`*`/`, where *`userid`* is your andrew ID.

11 Late Policy

- If you hand in on time, there is no penalty.
- 0-24 hours late = 25% penalty.
- 24-48 hours late = 50% penalty.
- More than 48 hours late = you lose all the points for this project.

NOTE: You can use your grace-days quota. For details about the quota, please refer to the syllabus.