15-440: Project 3
Clustering Data Points and DNA Strands Using MapReduce and MPI

Project posted on: October 24, 2011
Interim design report due date: October 31, 2011
Final due date: November 14, 2011

Intended Learning Outcomes

This project applies the theory of two popular programming models, Message Passing Interface (MPI) and MapReduce. The learning outcomes of the project are two-fold:

1. Apply MPI and MapReduce to a popular real problem, namely cluster analysis using K-Means algorithm.
2. Compare and contrast MPI-based and MapReduce-based implementations of K-Means algorithm in terms of performance and development effort.

Project Objectives

The overall goal of this project is to get a clear understating on how different parallel implementations for the same algorithm using different programming models can provide different performance and entail different development effort. Besides, students will conduct and analyze some scalability studies on various degrees of parallelism and data set sizes. Lastly, the reason behind exposing students to a real common clustering problem is due to the significance and importance of cluster analysis in various domains including, but not limited to, data mining and statistical data analysis. For whatever domain our student will be in, the chances are that sooner or later they will run into a clustering problem. This 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 popular programming models.

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 summary, clustering has to define some notion of “similarity” among 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 firstly made (see Fig. 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 Fig. 1(b)). The algorithm keeps iterating until the recomputed means (almost) stop varying (see Fig. 1(c)).

![Initial Means](image1)
![Recalculated Means](image2)
![Final Clusters](image3)

Fig. 1
In this project we will apply K-Means clustering to two different applications, data points in a 2D plane and DNA strands in biology.

**Clustering Data Points**

Considering a case of a data set composed of data points in \( d \)-dimensional space \( \mathbb{R}^d \). 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 \( \mathbb{R}^d \), 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:

Initialize \( m_i \) to \( k \) random \( x^d \), for \( i = 1, \ldots, k \) and \( x^d \in X \) that contains each of our \( d \)-dimensional data point.

Repeat

For all \( x^d \) in \( X \)

\[
\begin{align*}
b_i^d & \leftarrow 1 \text{ if } ||x^d - m_i|| = \min_j ||x^d - m_j|| \\
b_i^d & \leftarrow 0 \text{ otherwise}
\end{align*}
\]

For all \( m_i, i = 1, \ldots, k \)

\[
m_i \leftarrow \frac{\sum b_i^d x^d}{\sum b_i^d}
\]

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 $\mu$), essentially until the centroids start varying very little.

The positive integer $\mu$ is known as number of K-Means iterations. The precise value of $\mu$ 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.

**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 of 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 $S_1$ and $S_2$ is denoted as $F(S_1, S_2)$ and is equal to 18. On the other hand, $F(S_1, S_3) = 16$. The K-Means algorithm, described in the previous section, can be applied to DNA strands with this given similarity function $F(\cdot, \cdot, \cdot)$ 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.

**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 (both MPI-1 and MPI-2). Second, you will provide also a *MapReduce-based* version using Apache’s MapReduce implementation – Hadoop.

In addition, you have to write your own data set generator that generates a random number $P$ of DNA strands per cluster for $k$ clusters (use any programming language you like).
We will provide you with the following:

- A cluster of 4 VMs, each with 1 vCPU, 1 GB RAM, 20 GB storage and Fedora 15 64-Bit operating system.
- Both, Hadoop 0.20.2 and MPICH2 set up and ready on your cluster.
- A data set generator that generates a random set of 2D data points.

Your sequential, MPI-based, and MapReduce-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.

**Experimentation and Analysis**

Please conduct and provide the following:

- A comparison between your 3 different K-Means implementations in terms of performance and development effort.

- Two scalability studies on:

  - The number of processes for your MPI version with a fixed data set of 2D data points. Specifically, use 2, 4, 8, and 12 processes.
  - The number of data points in your data points’ data set with a fixed number of processes (say 8) and 1 reducer for your MPI and MapReduce applications, respectively. Specifically, use 10 million, 20 million, and 30 million data points.

- A discussion on:

  - Your experience in applying MPI and MapReduce to K-Means clustering algorithm.
  - Your insights concerning the performance trade-offs of MPI and MapReduce with K-Means.
  - Your thoughts on the applicability of K-Means to MPI and MapReduce.
• Your recommendations regarding the usage of MPI and MapReduce for algorithms similar to K-means.

**Deliverables**

**Design Document Deliverables**
Interim design document is due on October 31, 2011. The document should present the pseudo-codes of your parallel K-Means algorithm tailored for MPI and MapReduce. Clearly identify and describe your parallel model for solving the problem (i.e., Single Program Multiple Data (SPMD) or Multiple Program Multiple Data (MPMD) model). Lastly, please clearly present the steps involved in parallelizing K-Means using a program flow diagram. Detailed illustration of the flow diagram is mandatory.

**Final Deliverables**

As final deliverables, you should submit:

1. An archive containing a fully tested and debugged code for your data set generator, sequential, MPI and MapReduce K-Means versions.

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

**Handing In the Project**

Submit your documents and code on AFS directory:

/afs/qatar.cmu.edu/msakr/www/15440-f11/handin/userid/p3/

where ‘userid’ is your andrew user ID.
Late Policy

- If you hand in on time, there is no penalty (duh!).
- 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 grace-days quota, please read the syllabus.

Team Project Policy

This project is a team project. We will split you into teams of two students. Each team should divide the work, including design, coding, testing, analysis, write-up, and presentation, equally among all its members. Members of a team might get different grades on the project if after evaluation we recognize that the efforts put by members clearly vary. After submitting the project, each team has to schedule an appointment with one of the course instructors in order to present the delivered work.