15-440: Project 4

Characterizing MapReduce Task Parallelism using K-Means on the Cloud

School of Computer Science

Carnegie Mellon University, Qatar

Fall 2016

Assigned Date: November 15th, 2016

Due Date: November 30th, 2016

1 Summary & Intended Learning Outcomes

MapReduce is now a pervasive analytics engine on the cloud. Hadoop is an open source implementation of MapReduce and is currently enjoying wide popularity.

Hadoop offers a high-dimensional space of configuration parameters, which makes it difficult for practitioners to set for efficient and cost-effective execution. In this project, you will implement the *K-Means* clustering algorithm using Hadoop MapReduce and "characterize" the map phase parallelism accordingly. For the sake of this project, map phase parallelism is defined in terms of two Hadoop configurable parameters; the number of available map slots and the number of map tasks running over the slots. Within the confines of MapReduce (and this project), we define the characterization of the map phase parallelism as the process of observing, identifying and explaining various MapReduce runtime responses to different map task and map slot numbers. To this end, in addition to characterizing the map phase parallelism, you will compare and contrast the performance of your MPI *K-Means* implementation (from P3) against your Hadoop MapReduce *K-Means* implementation from this project.

In short, the learning outcomes of this project are as follows:

1. Apply MapReduce to a popular real problem, namely cluster analysis using *K-Means* algorithm.
2. Characterize the map phase parallelism using the *K-Means* clustering algorithm.
3. Compare and contrast your MPI and Hadoop MapReduce implementations of *K-Means* in terms of performance and development effort.

2 Project Objectives

The overall goal of this project is to get a clear understanding on Hadoop map parallelism and how different parallel implementations of the same algorithm compare against each other. You will conduct and analyze some scalability studies on various degrees of parallelism for *K-Means*. The project will provide students with: **(1)** deep insights onto how parallelism affects performance in large-scale settings, and **(2)** a practical experience augmented with a methodology for solving clustering problems (and alike) on a distributed system using MapReduce.

3 Implementation Guidelines

In this project, you will provide a MapReduce implementation for *K-Means* with two types of data sets; a data set of data points and a data set of DNA strands (same as in P3). Please use the datasets you generated in P3 to run and test your MapReduce *K-means* implementation. For a complete explanation of the K-Means algorithm, please refer to the write-up of P3.

For this project, use the 4-VM virtual cluster (which can be deemed as a private cloud) provided to you in P3. We have already installed and tested Hadoop 0.20.2 for you. Hence, the clusters are ready to run your Hadoop MapReduce code.

4 Experimentation & Analysis

After implementing *K-Means* using Hadoop MapReduce, please conduct experiments and report on the following:

- A comparison between your 3 different *K-Means* implementations (the sequential and the MPI ones from P3, and the MapReduce one from this project) in terms of performance and development effort.
- Three scalability studies. In particular:
  - A scalability study on the number of map slots per each virtual machine (VM) for fixed cluster, data set and HDFS block sizes. Specifically, use 1, 2, and 4 map slots per each VM on your 4-VM cluster, with a data set size of your choice (use only the data set of the 2D data points) and 64MB HDFS block size.
  - A scalability study on the HDFS block size per fixed data set size, cluster size and map slot value. Specifically, use 32MB, 64MB and 128MB HDFS block sizes on your 4-VM cluster with 2 map slots per each VM and a data set size of your choice (again, use only the data set of the 2D data points).
  - A scalability study on the number of VMs per fixed data set size, HDFS block size, and map slot value. Specifically, use 1, 2, 3 and 4 VMs with 64MB HDFS block size, 2 map slots per a VM and a data set size of your choice (again, use only the data set of the 2D data points).

    Note that these scalability studies are, in essence, the methodology we are using to characterize the map phase parallelism. You should report on your observations concerning the obtained parallelism (in each of the scalability studies) and the consequent overall influence on Hadoop MapReduce performance for *K-Means*. For instance, you should explain why a larger number of map tasks can sometimes expedite performance while at other times it might degrade performance.
- A discussion on:
  - Your experience in applying MapReduce to the *K-Means* clustering algorithms.
  - Your insights concerning the performance trade-offs of MPI and MapReduce with *K-Means*.
  - Your thoughts on the applicability of *K-Means* to MapReduce.
  - Your recommendations regarding the usage of MapReduce for algorithms similar to *K-Means*

5 Final Deliverables

As final deliverables, you should submit:

1. An archive containing a fully tested and debugged code for your MapReduce *K-Means* implementation. Specifically, you must submit the following:

   (a) ***Two programs for the K-Means implementation (one for the 2D data-set and one for the DNA strands data-set).***

   - Input:
     - The **Input directory** containing the data
     - The **Output directory** containing the data
     - Total number of **points/strands** provided
     - Number of **clusters**
     - Number of **iterations** *(if it's given as **0**, 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, and 5 iterations
       hadoop jar 2DPoints.jar 2DPoints /user/hadoop/2D/input
                 /user/hadoop/2D/output 100 3 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):

         ```
         // x-coordinate, y-coordinate, number of points
         5.212, 9.880, 400
         1.511, 3.201, 320
         ...
         ```

         Note: *that the output file can either be on HDFS or on the local directory of the JAR program. In case it's the former, please make it clear where exactly it will be stored.*

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

6 Handing In the Project

Submit your code using the AFS file system:
/afs/qatar.cmu.edu/usr10/mhhammou/www/15440-f16/handin/p4/*userid*/,
where *userid* is your Andrew ID.

7 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 **CANNOT** use your grace-days quota. For details about the grace-days quota, please refer to the course syllabus.