# 

# 

**TRƯỜNG ĐẠI HỌC THỦY LỢI**

**KHOA CÔNG NGHỆ THÔNG TIN**

--🙢🕮🙠--

A blue and white logo

Description automatically generated

**BÁO CÁO BÀI TẬP LỚN**

**ỨNG DỤNG THUẬT TOÁN K-MEAN TRÊN HADOOP**

**ĐỂ PHÂN CỤM GIÁ ĐIỆN THOẠI**

Giáo viên hướng dẫn: **TS. Lê Đắc Phương Thảo**

Sinh viên thực hiện: Nguyễn Minh Hiếu - 2151160536

Nguyễn Thanh Hưng - 2151060268

Nguyễn Thành Hưng - 2151160521

Nguyễn Hoàng Anh - 2151160491

NHÓM: 13

LỚP: 63HTTT2

**Hà Nội, tháng 11/2024**

**MỤC LỤC**

[CHƯƠNG I : GIỚI THIỆU 3](#_Toc181130304)

[1.1 Khái quát vấn đề 3](#_Toc181130305)

[1.2 Phạm vi 3](#_Toc181130306)

[CHƯƠNG 2: MỘT SỐ KIẾN THỨC CƠ BẢN 3](#_Toc181130307)

[2.1 Giới thiệu về dữ liệu lớn 4](#_Toc181130308)

[2.2 Tìm hiểu về Hadoop và MapReduce 5](#_Toc181130309)

[2.2.1 Hadoop 5](#_Toc181130310)

[2.2.2 Kiến trúc Hadoop 5](#_Toc181130311)

[2.2.3 MapReduce 6](#_Toc181130312)

[2.3 Giới thiệu về thuật toán K-Means 9](#_Toc181130313)

[CHƯƠNG 3. PHÂN CỤM DỮ LIỆU VỚI GIẢI THUẬT K-MEANS SONG SONG TRÊN MÔ HÌNH MAPREDUCE 10](#_Toc181130316)

[4.1 Bước 1: Tạo file chứa dữ liệu 11](#_Toc181130318)

[4.2 Bước 2: Tạo các class xử lý nhiệm vụ phân cụm K-means 12](#_Toc181130319)

[4.1 Bước 3: Tạo file jar 28](#_Toc181130326)

[4.3 Bước 4: Thử nghiệm trên file data-kmean.txt 32](#_Toc181130327)

[CHƯƠNG 5: KẾT LUẬN 34](#_Toc181130331)

[TÀI LIỆU THAM KHẢO 35](#_Toc181130332)

# CHƯƠNG I : GIỚI THIỆU

# Khái quát vấn đề

Bài toán tìm kiếm giá điện thoại là một trong những bài toán phổ biến của cuộc sống thường ngày. Khi mua hoặc bán một chiếc điện thoại, người mua hoặc hoặc người bán thường quan tâm đến giá trị thực tế của chiếc điện thoại đó để đảm bảo tính công bằng và hợp lý trong giao dịch.

Việc xác định giá trị thực tế của một chiếc điện thoại là một việc không phải dễ dàng. Nhiều yếu tố khác nhau có thể ảnh hưởng đến giá trị của một chiếc điện thoại, bao gồm nhà sản xuất, dòng máy, tuổi đời, tình trạng hiện tại,...và một vài yếu tố khác.

# Phạm vi

Phân cụm dữ liệu là một phương thức khai phá dữ liệu quan trọng. Việc phân tích gom cụm là tìm hiểu giải thuật và phương thức phân loại đối tượng. Một cụm là tập các đối tượng tương tự nhau hoặc là tập các thực thể hoặc nhóm các đối tượng mà các thực thể trong cùng cụm phải giống nhau; các thực thể ở các cụm khác nhau thì không giống nhau. Mỗi thực thể có thể có nhiều thuộc tính hoặc tính năng giống nhau được đo sự giống nhau dựa trên các thuộc tính hoặc tính năng giống nhau. Cùng với việc phát triển nhanh chóng của Internet, một lượng lớn tài liệu cần được xử lý trong một thời gian ngắn. Việc lưu trữ và tính toán dữ liệu lớn trong hệ thống phân tán là một phương thức đang được triển khai. Trong tính toán phân tán thì cần chia nhỏ công việc để mỗi công việc được xử lý trên một máy tính. MapReduce là mô hình lập trình song song bắt nguồn từ lập trình chức năng và được Google đề xuất để xử lý lượng dữ liệu lớn trong môi trường phân tán. Dự án Hadoop cung cấp hệ thống file phân tán (HDFS) và hỗ trợ mô hình MapReduce. Hadoop cho phép các ứng dụng làm việc với hàng ngàn nodes với hàng petabyte dữ liệu. Trong bài báo này, chúng tôi đề cập đến việc triển khai giải thuật gom cụm K-means trên mô hình lập trình song song MapReduce dựa trên nền tảng Hadoop [4] và đưa ra một số kết quả thử nghiệm gom cụm phân tán dựa trên mô hình này

# CHƯƠNG 2: MỘT SỐ KIẾN THỨC CƠ BẢN

# Giới thiệu về dữ liệu lớn

Theo wikipedia, Dữ liệu lớn là một thuật ngữ chỉ bộ dữ liệu lớn hoặc phức tạp mà các phương pháp truyền thống không đủ các ứng dụng để xử lý dữ liệu này.

Theo Gartner, Dữ liệu lớn là những nguồn thông tin có đặc điểm chung khối lượng lớn, tốc độ nhanh và dữ liệu định dạng dưới nhiều hình thức khác nhau, do đó muốn khai thác được đòi hỏi phải có nhiều hình thức xử lý mới để đưa ra quyết định, khám phá và tối ưu hóa quy trình.

Các đặc trưng cơ bản của dữ liệu lớn:

A diagram of various colored circles

Description automatically generated

Big Data là một phần thiết yếu của hầu hết mọi tổ chức ngày nay và để có được kết quả cần thiết thông qua phân tích những dữ liệu này, cần có một bộ công cụ ở mỗi giai đoạn để xử lý và phân tích dữ liệu. Khi lựa chọn bộ công cụ cần phải xem xét một số yếu tố như sau: kích thước của bộ dữ liệu, giá của công cụ, loại phân tích sẽ được thực hiện,…

Với sự tăng trưởng theo cấp số nhân của Big Data, thị trường tràn ngập với hàng loạt công cụ khác nhau. Những công cụ giúp giảm thiểu chi phí và do đó làm tăng tốc độ phân tích: Apache Hadoop, Apache Spark, Apache Storm, Apache Cassandra, MongoDB, …

# Tìm hiểu về Hadoop và MapReduce

# Hadoop

Hadoop là một framework giúp lưu trữ và xử lý Big Data áp dụng MapReduce. Nói đơn giản cách khác nó sẽ giúp sắp xếp dữ liệu sao cho user có thể dễ dàng sử dụng nhất.

MapReduce được Google tạo ra ban đầu để xử lý đống dữ liệu lớn của công ty họ. Ta còn có thể gọi phương pháp này là Phân tán dữ liệu vì nó tách hết tập hợp các dữ liệu ban đầu thành các dữ liệu nhỏ và sắp xếp lại chúng để dễ dàng tìm kiếm và truy xuất hơn, đặc biệt là việc truy xuất các dữ liệu tương đồng. Ví dụ thường thấy nhất là các đề xuất mà ta hay thấy ở Google tìm kiếm

Như vậy mô hình lập trình MapReduce là nền tảng ý tưởng của Hadoop. Bản thân Hadoop là một framework cho phép phát triển các ứng dụng phân tán phần cứng thông thường . Các phần cứng này thường có khả năng hỏng hóc cao. Khác với loại phần cứng chuyên dụng đắt tiền, khả năng xảy ra lỗi thấp như các supermicrocomputer chẳng hạn.

Hadoop viết bằng Java. Tuy nhiên, nhờ cơ chế streaming, Hadoop cho phép phát triển các ứng dụng phân tán bằng cả java lẫn một số ngôn ngữ lập trình khác như C++, Python, Perl.

# Kiến trúc Hadoop

Hadoop gồm 4 module:

* + - * Hadoop Common: Đây là các thư viện và tiện ích cần thiết của Java để các module khác sử dụng. Những thư viện này cung cấp hệ thống file và lớp OS trừu tượng, đồng thời chứa các mã lệnh Java để khởi động Hadoop.
      * Hadoop YARN: Đây là framework để quản lý tiến trình và tài nguyên của các cluster.
      * Hadoop Distributed File System (HDFS): Đây là hệ thống file phân tán cung cấp truy cập thông lượng cao cho ứng dụng khai thác dữ liệu.

A diagram of a computer network

Description automatically generated

* + - * Hadoop MapReduce: Đây là hệ thống dựa trên YARN dùng để xử lý song song các tập dữ liệu lớn.

Hiện nay Hadoop đang ngày càng được mở rộng cũng như được nhiều framework khác hỗ trợ như Hive, Hbase, Pig. Tùy vào mục đích sử dụng mà ta sẽ áp dụng framework phù hợp để nâng cao hiệu quả xử lý dữ liệu của Hadoop

# MapReduce

Mapreduce là một mô hình được Google thiết kế độc quyền với khả năng lập trình xử lý một lượng lớn các dữ liệu song song đồng thời phân tán các thuật toán trên cùng một máy tính. Mặc dù Mapreduce ban đầu là một công nghệ độc quyền của Google nhưng trong thời gian gần đây, Mapreduce đang dần trở thành một trong những thuật ngữ tổng quát hoá.

Mapreduce sẽ bao gồm hai thủ tục (giai đoạn) chính là Map và Reduce:

+ Thủ tục Map bao gồm bộ lọc (filter) và phân loại (sort) trên dữ liệu.

+ Thủ tục Reduce sẽ thực hiện quá trình tổng hợp toàn bộ dữ liệu.

Mô hình Mapreduce được thiết kế dựa trên các khái niệm biến đổi của một bản đồ và thiết lập các chức năng lập trình đi theo hướng chức năng. Thư viện của thủ tục Map và thủ tục Reduce sẽ được viết bằng đa dạng các loại ngôn ngữ lập trình khác nhau. Các thủ tục này sẽ được cài đặt hoàn toàn miễn phí và Apache Hadoop là thủ tục MapReduce được sử dụng phổ biến nhất.

Ưu điểm của MapReduce:

+ Mapreduce có thể dễ dàng xử lý tất cả mọi bài toán có lượng dữ liệu khổng lồ nhờ khả năng tính toán và tác vụ phân tích phức tạp.

+ Mapreduce có khả năng chạy song song trên các máy tính có sự phân tán khác nhau với khả năng hoạt động độc lập kết hợp với việc phân tán và xử lý các lỗi kỹ thuật để mang đến hiệu quả cao cho toàn bộ hệ thống.

+ Mapreduce có khả năng thực hiện được trên đa dạng nhiều loại ngôn ngữ lập trình khác nhau như ngôn ngữ C/C++, Java, Perl, Python, Ruby,... cùng với những thư viện hỗ trợ tương ứng.

+ Mã độc trên Internet ngày càng nhiều khiến cho việc xử lý các đoạn mã độc này trở nên phức tạp và tiêu tốn nhiều thời gian hơn. Do đó, Mapreduce đang dần hướng quan tâm nhiều hơn cho việc phát hiện các mã độc để có thể nhanh chóng xử lý các đoạn mã độc đó. Nhờ đó, hệ điều hành được đảm bảo vận hành trơn tru với tính bảo mật cao nhất.

A diagram of a diagram

Description automatically generated

Bộ ánh xạ (Mapper): xử lý tập dữ liệu đầu vào dưới dạng (key, value) và tạo ra tập dữ liệu trung gian là cặp (key, value). Tập dữ liệu này được tổ chức cho hoạt động của Reduce. Bộ ánh xạ Map thực hiện các bước như sau:

Bước 1: Ánh xạ cho mỗi nhóm dữ liệu đầu vào dưới dạng (key, value).

Bước 2: Thực thi việc Map, xử lý cặp (key, value) để tạo (key, value) mới, công việc này được gọi là chia nhóm, tức là tạo các giá trị liên quan tương ứng với cùng các từ khóa.

Bước 3: Đầu ra của bộ ánh xạ được lưu trữ và định vị cho mỗi bộ Reducer. Tổng các khối và số công việc reduce là như nhau.

Bộ Reducer: Trộn tất cả các phần tử value có cùng key trong tập dữ liệu trung gian do Map tạo ra để tạo thành tập trị nhỏ hơn và quá trình này được lặp lại cho tất cả các giá trị key khác nhau. Việc truyền dữ liệu được thực hiện giữa Map và Reduce. Lập trình viên cần cài đặt chính xác (key, value), MapReduce sẽ gom cụm một cách tự động và chính xác các values với cùng key. Các bước của bộ Reducer:

Bước 1 (Trộn): Đầu vào của Reducer là đầu ra của Mapper. Giai đoạn này, MapReduce sẽ gán khối liên quan cho bộ Reducer.

Bước 2 (Sắp xếp): Giai đoạn này, đầu vào của bộ Reducer được gom theo key (do đầu ra của bộ ánh xạ khác nhau có thể có cùng key). Hai giai đoạn Trộn và sắp xếp được đồng bộ hóa.

Bước 3: Tạo bộ so sánh để nhóm các keys trung gian lần hai nếu quy luật gom nhóm khác với trước đó.

Trong một tiến trình của MapReduce, các công việc Map chạy song song, các công việc Reduce chạy song song. Tuy nhiên, các công việc Map và Reduce

được thực hiện tuần tự, tức là Map phải hoàn thành và chuyển dữ liệu cho Reduce. Dữ liệu đầu vào và đầu ra được lưu trữ trong hệ thống file.

# Giới thiệu về thuật toán K-Means

K-Means là thuật toán rất quan trọng và được sử dụng phổ biến trong kỹ thuật phân cụm. Tư tưởng chính của thuật toán K-Means là tìm cách phân nhóm các đối tượng (objects) đã cho vào K cụm (K là số các cụm được xác định trước, K nguyên dương) sao cho tổng bình phương khoảng cách giữa các đối tượng đến tâm nhóm (centroid) là nhỏ nhất.

Thuật toán K-Means thực hiện qua các bước chính sau:

Bước 1: Chọn ngẫu nhiên K tâm (centroid) cho K cụm (cluster). Mỗi cụm được đại diện bằng các tâm của cụm.

Bước 2: Tính khoảng cách giữa các đối tượng (objects) đến K tâm (thường dùng khoảng cách Euclidean)

Bước 3: Nhóm các đối tượng vào nhóm gần nhất

Bước 4: Xác định lại tâm mới cho các nhóm

Bước 5: Thực hiện lại bước 2 cho đến khi không có sự thay đổi nhóm nào của các đối tượng

# Sơ đồ thuật toán

# A diagram of a diagram Description automatically generated

# CHƯƠNG 3. PHÂN CỤM DỮ LIỆU VỚI GIẢI THUẬT K-MEANS SONG SONG TRÊN MÔ HÌNH MAPREDUCE

# A diagram of a map task Description automatically generated

Giải thuật K-Means dựa trên mô hình MapReduce làm việc như sau:

* Bước 1: Giai đoạn đầu là tiền xử lý tài liệu. Chia tập tài liệu D thành m tập con. Có hai công việc MapReduce trong giai đoạn này, đầu tiên là phải tính toán các tham số cho bước tiếp theo như đếm từ, tính TF,.., tiếp theo là tính TFIDF (con số thể hiện mức độ quan trọng của từ trong một tài liệu trên tập các tài liệu) của mỗi tài liệu. Kết thúc giai đoạn này tài liệu được đánh chỉ số cũng như vector trọng số của nó cũng đã hoàn chỉnh, đã chọn được k tài liệu làm tâm, xác định kích thước mảnh dữ liệu, ngưỡng hội tụ để chương trình kết thúc.
* Bước 2: Giai đoạn thứ hai là hàm map, đọc dữ liệu đầu vào và tính khoảng cách tới mỗi tâm. Với mỗi tài liệu, nó tạo ra cặp . Rất nhiều dữ liệu được tạo ra trong giai đoạn này, hàm gộp có thể được sử dụng để giảm kích thước trước khi gửi đến Reduce.
* Bước 3: Giai đoạn thứ 3 là hàm reduce, hàm này tính tọa độ mới cho tâm các cụm. Dữ liệu đầu ra này được ghi vào tập tin của cụm bao gồm: số lần lặp, id cụm, các tọa độ của tâm cụm, kích thước của cụm.
* Bước 4: Cuối cùng các tọa độ của cụm mới được so sánh với các tọa độ ban đầu. Nếu hàm điều kiện hội tụ thì chương trình chuyển sang bước 5 và ta tìm được các cụm. Nếu không, sử dụng các tâm của cụm mới thực hiện và lặp lại từ bước 2 đến bước 4.
* Bước 5: Lấy các tâm cụm từ JOB 1. Tính khoảng cách từ từng điểm đến từng tâm cụm, dựa vào khoảng cách ngắn nhất phân cụm điểm đó. In ra kết quả các tâm cụm và các điểm thuộc từng cụm.

**CHƯƠNG 4: THỰC NGHIỆM**

# Bước 1: Tạo file chứa dữ liệu

Tạo file data-kmeans.txt với nội dung gồm:

Cột 1: popularity là độ phổ biến của chiếc điện thoại (Trong khoảng từ 1 đến 1224)

Cột 2: best\_price là giá tiền tốt nhất cho một chiếc điện thoại ứng với độ phổ biến

# Bước 2: Tạo các class xử lý nhiệm vụ phân cụm K-means

# Double click vào project KmeansMapReduce, chuột phải vào src và chọn New > Class

# A screenshot of a computer Description automatically generated

# Nội dung bên trong file KMapper.java:

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class KMapper extends Mapper<LongWritable, Text, LongWritable, PointWritable> {

private PointWritable[] currCentroids;

private final LongWritable centroidId = new LongWritable();

private final PointWritable pointInput = new PointWritable();

@Override

public void setup(Context context) {

int nClusters = Integer.parseInt(context.getConfiguration().get("k"));

this.currCentroids = new PointWritable[nClusters];

for (int i = 0; i < nClusters; i++) {

String[] centroid = context.getConfiguration().getStrings("C" + i);

*// this.currCentroids[i] = new PointWritable(centroid[0].split(","));*

this.currCentroids[i] = new PointWritable(centroid);

}

}

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

String[] arrPropPoint = value.toString().split(",");

pointInput.set(arrPropPoint);

double minDistance = Double.MAX\_VALUE;

int centroidIdNearest = 0;

for (int i = 0; i < currCentroids.length; i++) {

System.out.println("currCentroids[" + i + "]=" + currCentroids[i].toString());

double distance = pointInput.calcDistance(currCentroids[i]);

if (distance < minDistance) {

centroidIdNearest = i;

minDistance = distance;

}

}

centroidId.set(centroidIdNearest);

context.write(centroidId, pointInput);

}

}

# Nội dung bên trong file KReducer.java:

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class KReducer extends Reducer<LongWritable, PointWritable, Text, Text> {

private final Text newCentroidId = new Text();

private final Text newCentroidValue = new Text();

public void reduce(LongWritable centroidId, Iterable<PointWritable> partialSums, Context context)

throws IOException, InterruptedException {

PointWritable ptFinalSum = PointWritable.copy(partialSums.iterator().next());

while (partialSums.iterator().hasNext()) {

ptFinalSum.sum(partialSums.iterator().next());

}

ptFinalSum.calcAverage();

newCentroidId.set(centroidId.toString());

newCentroidValue.set(ptFinalSum.toString());

context.write(newCentroidId, newCentroidValue);

}

}

# Nội dung bên trong file KCombiner.java:

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapreduce.Reducer;

public class KCombiner extends Reducer<LongWritable, PointWritable, LongWritable, PointWritable> {

public void reduce(LongWritable centroidId, Iterable<PointWritable> points, Context context)

throws IOException, InterruptedException {

PointWritable ptSum = PointWritable.copy(points.iterator().next());

while (points.iterator().hasNext()) {

ptSum.sum(points.iterator().next());

}

context.write(centroidId, ptSum);

}

}

# Nội dung bên trong file PointWritable:

import java.io.DataInput;

import java.io.DataOutput;

import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class PointWritable implements Writable {

private float[] attributes = null;

private int dim;

private int nPoints;

public PointWritable() {

this.dim = 0;

}

public PointWritable(final float[] c) {

this.set(c);

}

public PointWritable(final String[] s) {

this.set(s);

}

public static PointWritable copy(final PointWritable p) {

PointWritable ret = new PointWritable(p.attributes);

ret.nPoints = p.nPoints;

return ret;

}

public void set(final float[] c) {

this.attributes = c;

this.dim = c.length;

this.nPoints = 1;

}

public void set(final String[] s) {

this.attributes = new float[s.length];

this.dim = s.length;

this.nPoints = 1;

for (int i = 0; i < s.length; i++) {

this.attributes[i] = Float.parseFloat(s[i]);

}

}

@Override

public void readFields(final DataInput in) throws IOException {

this.dim = in.readInt();

this.nPoints = in.readInt();

this.attributes = new float[this.dim];

for (int i = 0; i < this.dim; i++) {

this.attributes[i] = in.readFloat();

}

}

@Override

public void write(final DataOutput out) throws IOException {

out.writeInt(this.dim);

out.writeInt(this.nPoints);

for (int i = 0; i < this.dim; i++) {

out.writeFloat(this.attributes[i]);

}

}

@Override

public String toString() {

StringBuilder point = new StringBuilder();

for (int i = 0; i < this.dim; i++) {

point.append(Float.toString(this.attributes[i]));

if (i != dim - 1) {

point.append(",");

}

}

return point.toString();

}

public void sum(PointWritable p) {

for (int i = 0; i < this.dim; i++) {

this.attributes[i] += p.attributes[i];

}

this.nPoints += p.nPoints;

}

public double calcDistance(PointWritable p) {

double dist = 0.0f;

for (int i = 0; i < this.dim; i++) {

dist += Math.pow(Math.abs(this.attributes[i] - p.attributes[i]), 2);

}

dist = Math.sqrt(dist);

return dist;

}

public void calcAverage() {

for (int i = 0; i < this.dim; i++) {

float temp = this.attributes[i] / this.nPoints;

this.attributes[i] = (float) Math.round(temp \* 100000) / 100000.0f;

}

this.nPoints = 1;

}

}

# Nội dung bên trong file Main:

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Date;

import java.util.List;

import java.util.Random;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.fs.FSDataOutputStream;

import org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class Main extends Configured implements Tool {

public static PointWritable[] initRandomCentroids(int kClusters, int nLineOfInputFile, String inputFilePath,

Configuration conf) throws IOException {

System.out.println("Initializing random " + kClusters + " centroids...");

PointWritable[] points = new PointWritable[kClusters];

List<Integer> lstLinePos = new ArrayList<Integer>();

Random random = new Random();

int pos;

while (lstLinePos.size() < kClusters) {

pos = random.nextInt(nLineOfInputFile);

if (!lstLinePos.contains(pos)) {

lstLinePos.add(pos);

}

}

Collections.sort(lstLinePos);

FileSystem hdfs = FileSystem.get(conf);

FSDataInputStream in = hdfs.open(new Path(inputFilePath));

BufferedReader br = new BufferedReader(new InputStreamReader(in));

int row = 0;

int i = 0;

while (i < lstLinePos.size()) {

pos = lstLinePos.get(i);

String point = br.readLine();

if (row == pos) {

points[i] = new PointWritable(point.split(","));

i++;

}

row++;

}

br.close();

return points;

}

public static void saveCentroidsForShared(Configuration conf, PointWritable[] points) {

for (int i = 0; i < points.length; i++) {

String centroidName = "C" + i;

conf.unset(centroidName);

conf.set(centroidName, points[i].toString());

}

}

public static PointWritable[] readCentroidsFromReducerOutput(Configuration conf, int kClusters,

String folderOutputPath) throws IOException, FileNotFoundException {

PointWritable[] points = new PointWritable[kClusters];

FileSystem hdfs = FileSystem.get(conf);

FileStatus[] status = hdfs.listStatus(new Path(folderOutputPath));

for (int i = 0; i < status.length; i++) {

if (!status[i].getPath().toString().endsWith("\_SUCCESS")) {

Path outFilePath = status[i].getPath();

System.out.println("read " + outFilePath.toString());

BufferedReader br = new BufferedReader(new InputStreamReader(hdfs.open(outFilePath)));

String line = null;*// br.readLine();*

while ((line = br.readLine()) != null) {

System.out.println(line);

String[] strCentroidInfo = line.split("\t"); *// Split line in K,V*

int centroidId = Integer.parseInt(strCentroidInfo[0]);

String[] attrPoint = strCentroidInfo[1].split(",");

points[centroidId] = new PointWritable(attrPoint);

}

br.close();

}

}

hdfs.delete(new Path(folderOutputPath), true);

return points;

}

private static boolean checkStopKMean(PointWritable[] oldCentroids, PointWritable[] newCentroids, float threshold) {

boolean needStop = true;

System.out.println("Check for stop K-Means if distance <= " + threshold);

for (int i = 0; i < oldCentroids.length; i++) {

double dist = oldCentroids[i].calcDistance(newCentroids[i]);

System.out.println("distance centroid[" + i + "] changed: " + dist + " (threshold:" + threshold + ")");

needStop = dist <= threshold;

*// chỉ cần 1 tâm < ngưỡng thì return false*

if (!needStop) {

return false;

}

}

return true;

}

private static void writeFinalResult(Configuration conf, PointWritable[] centroidsFound, String outputFilePath,

PointWritable[] centroidsInit) throws IOException {

FileSystem hdfs = FileSystem.get(conf);

FSDataOutputStream dos = hdfs.create(new Path(outputFilePath), true);

BufferedWriter br = new BufferedWriter(new OutputStreamWriter(dos));

for (int i = 0; i < centroidsFound.length; i++) {

br.write(centroidsFound[i].toString());

br.newLine();

System.out.println("Centroid[" + i + "]: (" + centroidsFound[i] + ") init: (" + centroidsInit[i] + ")");

}

br.close();

hdfs.close();

}

public static PointWritable[] copyCentroids(PointWritable[] points) {

PointWritable[] savedPoints = new PointWritable[points.length];

for (int i = 0; i < savedPoints.length; i++) {

savedPoints[i] = PointWritable.copy(points[i]);

}

return savedPoints;

}

public static int MAX\_LOOP = 50;

public static void printCentroids(PointWritable[] points, String name) {

System.out.println("=> CURRENT CENTROIDS:");

for (int i = 0; i < points.length; i++)

System.out.println("centroids(" + name + ")[" + i + "]=> :" + points[i]);

System.out.println("----------------------------------");

}

public int run(String[] args) throws Exception {

Configuration conf = getConf();

String inputFilePath = conf.get("in", null);

String outputFolderPath = conf.get("out", null);

String outputFileName = conf.get("result", "result.txt");

int nClusters = conf.getInt("k", 3);

float thresholdStop = conf.getFloat("thresh", 0.001f);

int numLineOfInputFile = conf.getInt("lines", 0);

MAX\_LOOP = conf.getInt("maxloop", 50);

int nReduceTask = conf.getInt("NumReduceTask", 1);

if (inputFilePath == null || outputFolderPath == null || numLineOfInputFile == 0) {

System.err.printf(

"Usage: %s -Din <input file name> -Dlines <number of lines in input file> -Dout <Folder ouput> -Dresult <output file result> -Dk <number of clusters> -Dthresh <Threshold>\n",

getClass().getSimpleName());

ToolRunner.printGenericCommandUsage(System.err);

return -1;

}

System.out.println("---------------INPUT PARAMETERS---------------");

System.out.println("inputFilePath:" + inputFilePath);

System.out.println("outputFolderPath:" + outputFolderPath);

System.out.println("outputFileName:" + outputFileName);

System.out.println("maxloop:" + MAX\_LOOP);

System.out.println("numLineOfInputFile:" + numLineOfInputFile);

System.out.println("nClusters:" + nClusters);

System.out.println("threshold:" + thresholdStop);

System.out.println("NumReduceTask:" + nReduceTask);

System.out.println("--------------- STATR ---------------");

PointWritable[] oldCentroidPoints = initRandomCentroids(nClusters, numLineOfInputFile, inputFilePath, conf);

PointWritable[] centroidsInit = copyCentroids(oldCentroidPoints);

printCentroids(oldCentroidPoints, "init");

saveCentroidsForShared(conf, oldCentroidPoints);

int nLoop = 0;

PointWritable[] newCentroidPoints = null;

long t1 = (new Date()).getTime();

while (true) {

nLoop++;

if (nLoop == MAX\_LOOP) {

break;

}

Job job = new Job(conf, "K-Mean");*// Job thực hiện deepCopy conf*

job.setJarByClass(Main.class);

job.setMapperClass(KMapper.class);

job.setCombinerClass(KCombiner.class);

job.setReducerClass(KReducer.class);

job.setMapOutputKeyClass(LongWritable.class);

job.setMapOutputValueClass(PointWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, new Path(inputFilePath));

FileOutputFormat.setOutputPath(job, new Path(outputFolderPath));

job.setOutputFormatClass(TextOutputFormat.class);

job.setNumReduceTasks(nReduceTask);

boolean ret = job.waitForCompletion(true);

if (!ret) {

return -1;

}

newCentroidPoints = readCentroidsFromReducerOutput(conf, nClusters, outputFolderPath);

printCentroids(newCentroidPoints, "new");

boolean needStop = checkStopKMean(newCentroidPoints, oldCentroidPoints, thresholdStop);

oldCentroidPoints = copyCentroids(newCentroidPoints);

if (needStop) {

break;

} else {

saveCentroidsForShared(conf, newCentroidPoints);

}

}

if (newCentroidPoints != null) {

System.out.println("------------------- FINAL RESULT -------------------");

writeFinalResult(conf, newCentroidPoints, outputFolderPath + "/" + outputFileName, centroidsInit);

}

System.out.println("----------------------------------------------");

System.out.println("K-MEANS CLUSTERING FINISHED!");

System.out.println("Loop:" + nLoop);

System.out.println("Time:" + ((new Date()).getTime() - t1) + "ms");

return 1;

}

public static void main(String[] args) throws Exception {

int exitCode = ToolRunner.run(new Main(), args);

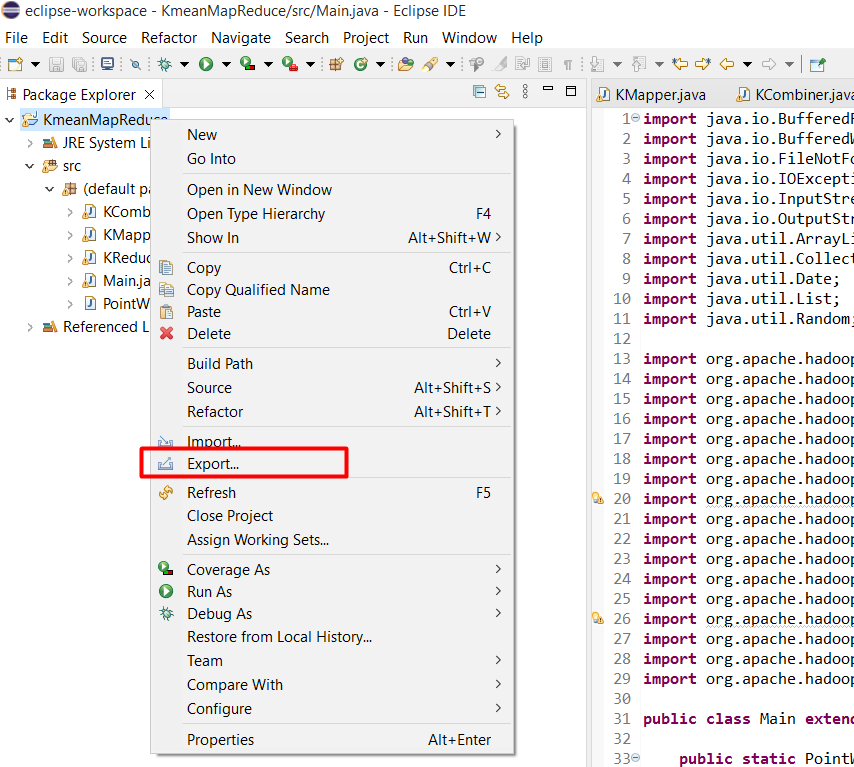
System.exit(exitCode);

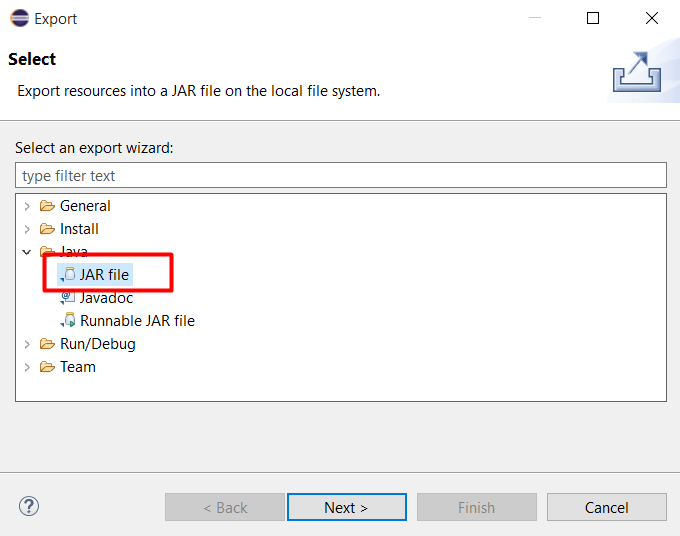
}

}

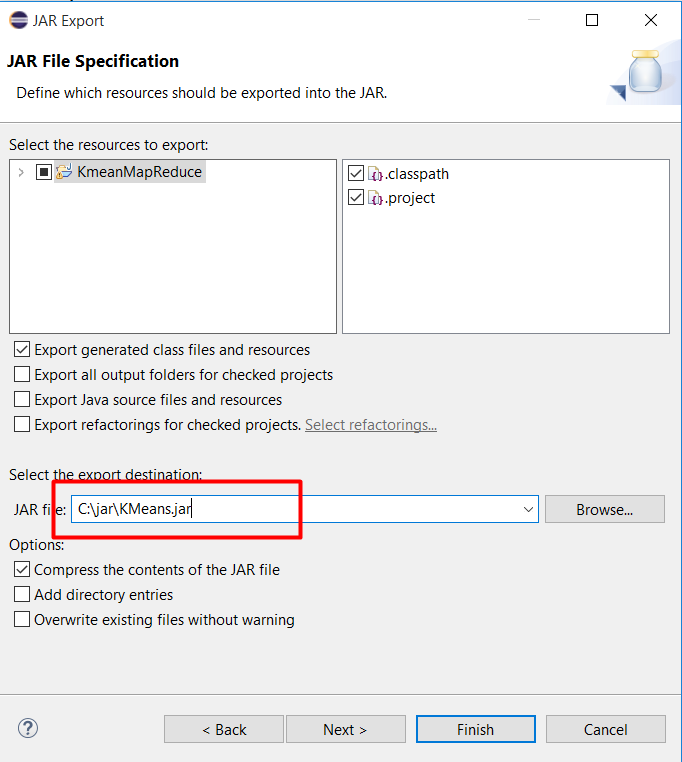
# Bước 3: Tạo file jar

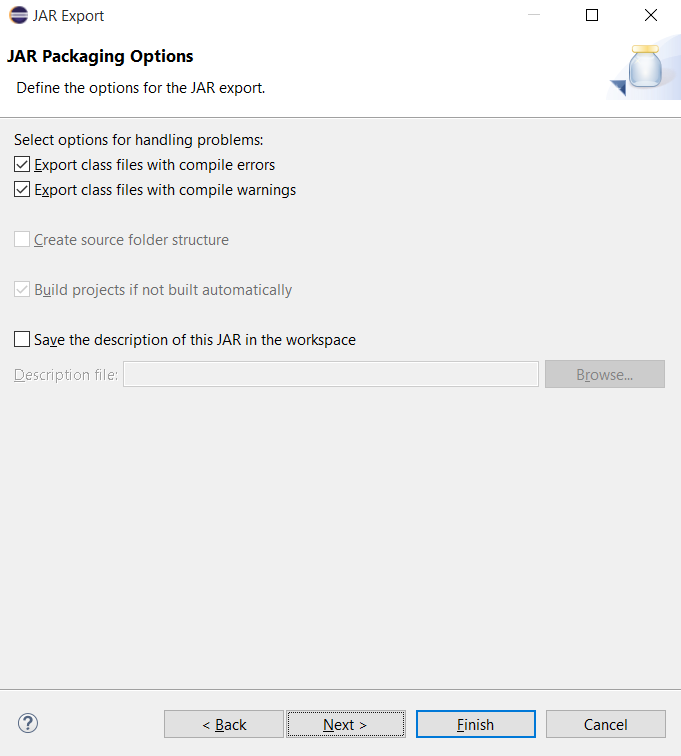
Chuột phải vào project **KmeanMapReduce** chọn **Export**



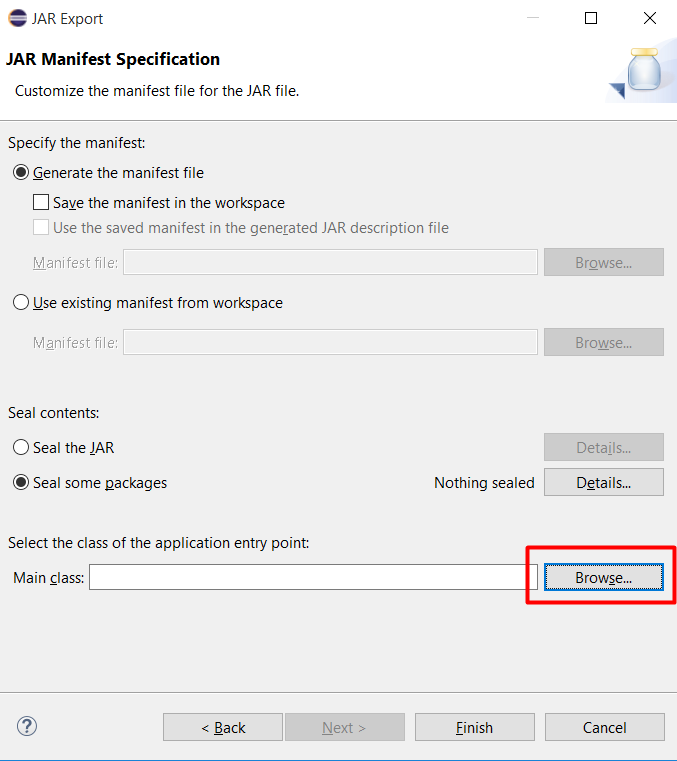
Chọn **Java > JAR File** rồi bấm **Next**

Chọn đường dẫn lưu file JAR và bấm **Next**

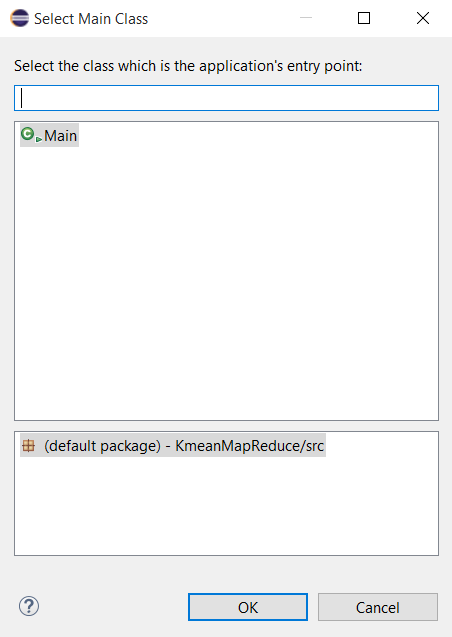


Bấm **Next**

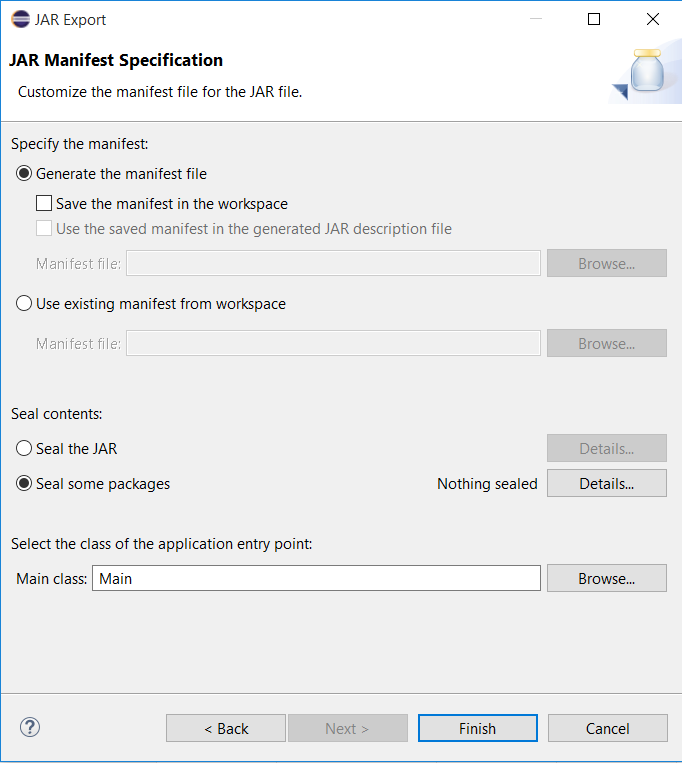
Bấm **Browser** để chọn file **main**



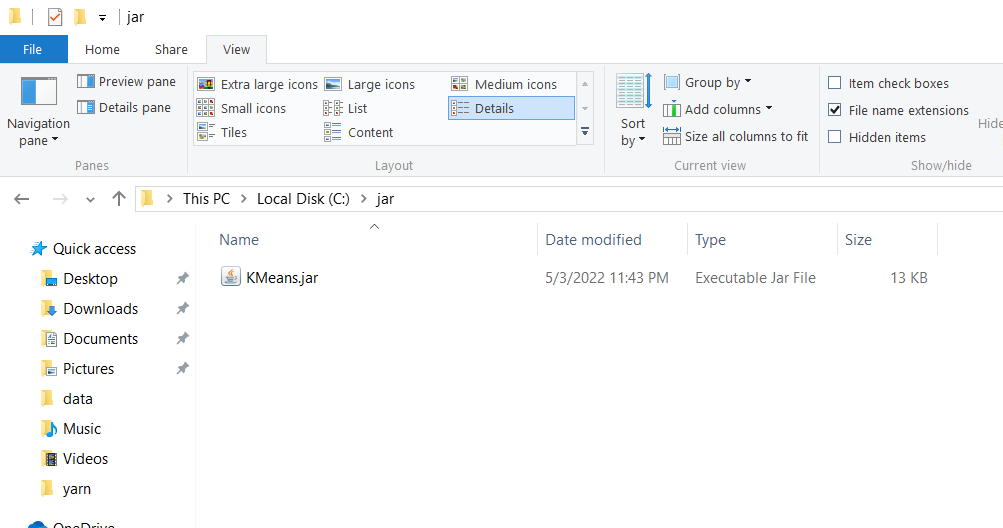
Chọn **Main** và bấm **OK**



Bấm **Finish** để thực hiện quá trình **Export**



Vào thư mục chứa lưu file JAR vừa tạo và kiểm tra kết quả



# Bước 4: Thử nghiệm trên file data-kmean.txt

Thử nghiệm trên file dữ liệu data-kmean.txt đã tạo ở trên, và kết quả thu được lưu tại result.txt trong thư mục k-output. Chạy lệnh sau:

*hadoop jar C:\jar\KMeans2.jar -Din /k-input1/data-kmean.txt -Dlines 30 - Dresult result.txt -Dmaxloop 5 -Dk 3 -Dthresh 0.0001 -DNumReduceTask*

*2 -Dout /k-output3*

# Minh họa input

# A screenshot of a computer Description automatically generated

# Output

A screenshot of a computer

Description automatically generated

# 

# CHƯƠNG 5: KẾT LUẬN

Trong bài tập lớn này, chúng em đã tìm hiểu và trình bày về kỹ thuật phân cụm K-Means cùng với ứng dụng Hadoop MapReduce cho bài toán phân cụm chất lượng nước.

Từ kết quả nghiên cứu các thuật toán K-Means, thuật toán ban đầu không thể thực hiện với tập dữ liệu lớn. Do đó, chúng em đã sử dụng mô hình MapReduce để phân cụm nước, với kỹ thuật này hiệu suất, thời lượng và tính ổn định của việc phân cụm dữ liệu lớn được cải thiện và hiệu quả hơn nhiều so với giải thuật K-Means ban đầu. Kết quả thực nghiệm cũng cho thấy tính hiệu quả của việc phát triển giải thuật K-Means trên mô hình MapReduce.

Sau bài tập lớn chúng em đã tiếp thu được nhiều kiến thực hữu ích để có thể phát triển bản thân trong môi trường học tập cũng như là môi trường công việc.

# TÀI LIỆU THAM KHẢO

[1] [http://it.husc.edu.vn/Media/ChuyenMuc/KhoaHoc/Hoithao-Hoinghi/SAICT\_20 15\_submission\_5.pdf](http://it.husc.edu.vn/Media/ChuyenMuc/KhoaHoc/Hoithao-Hoinghi/SAICT_20%2015_submission_5.pdf)

[2] <https://viblo.asia/p/tim-hieu-ve-hadoop-bJzKmOBXl9N>

[3] <https://insight.isb.edu.vn/top-8-cong-cu-big-data-ban-nen-biet/>

[4] [https://www.kaggle.com/datasets/artempozdniakov/ukrainian-market-mobile-](http://www.kaggle.com/datasets/artempozdniakov/ukrainian-market-mobile-) phones-data?fbclid=IwAR0bjy4LQTSy7TCUkjXrKfLFdmvqVJpGchcxMWWL9vo7KdSoYn0KGA6iezo