**TRƯỜNG ĐẠI HỌC THỦY LỢI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**----------🙡🕮🙣----------**



Nguyễn Minh Tuấn

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN PHÂN TÍCH DỮ LIỆU LỚN

ĐỀ TÀI:

ỨNG DỤNG THUẬT TOÁN K-MEANS VÀ MÔ HÌNH MAPREDUCE

ĐỂ PHÂN CỤM KHÁCH HÀNG

Người hướng dẫn: TS. Tạ Quang Chiểu

Hà Nội, ngày 13, tháng 04, năm 2023

MỤC LỤC

[LỜI CAM ĐOAN 4](#_Toc133930270)

[DANH MỤC CÁC HÌNH ẢNH 4](#_Toc133930271)

[Chương 1: Giới thiệu đề tài 4](#_Toc133930272)

[1.1 Giới thiệu 4](#_Toc133930273)

[1.2 Ngữ cảnh và lý do thực hiện đề tài 4](#_Toc133930274)

[1.2.1 Sự bùng phát dữ liệu và bài toán kinh doanh từ dữ liệu khách hàng 4](#_Toc133930275)

[1.2.2 Sự ra đời của mô hình Mapreduce 4](#_Toc133930276)

[1.2.3 Lý do thực hiện 4](#_Toc133930277)

[1.3 Nội dung báo cáo BÀI TẬP LỚN 4](#_Toc133930278)

[CHƯƠNG 2 NỀN TẢNG TÍNH TOÁN PHÂN TÁN VỚI HADOOP 4](#_Toc133930279)

[2.1 Giới thiệu Framework Hadoop 4](#_Toc133930280)

[2.1.1Hadoop là gì 4](#_Toc133930281)

[2.1.2 Lịch sử Hadoop 4](#_Toc133930282)

[2.1.3 Các thành phần của Hadoop 4](#_Toc133930283)

[2.1.4 Ứng dụng của Hadoop trong một số công ty 4](#_Toc133930284)

[2.1.5 Tổng quan của một Hadoop cluster 4](#_Toc133930285)

[2.2 Hadoop Distributed File System (HDFS) 4](#_Toc133930286)

[2.2.1 Giới thiệu 4](#_Toc133930287)

[2.2.2 Tổng quan thiết kế của HDFS 4](#_Toc133930288)

[2.2.3 Các tính năng của NameNode 4](#_Toc133930289)

[2.2.4 Khả năng chịu lỗi và chẩn đoán lỗi của HDFS 4](#_Toc133930290)

[2.2.5 Các giao diện tương tác 4](#_Toc133930291)

[2.2.6 Quản trị HDFS 4](#_Toc133930292)

[2.3 MapReduce 4](#_Toc133930293)

[2.3.1 Giới thiệu mô hình tính toán MapReduce 4](#_Toc133930294)

[2.3.2 Hadoop MapReduce Engine 4](#_Toc133930295)

[CHƯƠNG 3 Thuật toán phân cụm Kmean kết hợp mô hình Mapreduce 4](#_Toc133930296)

[3.1. Kmean là gì ? 4](#_Toc133930297)

[3.1.1 Học không giám sát 4](#_Toc133930298)

[3.1.2 Giới thiệu thuật toán Kmean 4](#_Toc133930299)

[3.1.3 Phân tích toán học 4](#_Toc133930300)

[3.1.4 Điều kiện dừng thuật toán 4](#_Toc133930301)

[3.15. Chỉ số đánh giá Davies-Bouldin 4](#_Toc133930302)

[3.2. Kmean sử dụng Mapreduce 4](#_Toc133930303)

[3.2.1 Nhận xét thuật toán Kmean 4](#_Toc133930304)

[3.2.2 Ý tưởng kết hợp mapreduce 4](#_Toc133930305)

[3.2.3 Giải pháp Mapreduce hóa Kmean 4](#_Toc133930306)

[3.2.4 Hàm map\_Kmean 4](#_Toc133930307)

[3.2.5 Hàm Reduce\_Kmean 4](#_Toc133930308)

[CHƯƠNG 4 Báo cáo kết quả thực nghiệm 4](#_Toc133930309)

[4.1 Thu thập và tiền xử lý dữ liệu 4](#_Toc133930310)

[4.2 Công cụ và thư viện sử dụng 4](#_Toc133930311)

[4.3 Thực nghiệm và đánh giá kết quả 4](#_Toc133930312)

[CHƯƠNG 5 Tổng kết bài báo cáo 4](#_Toc133930313)

[5.1 Tổng Kết nội dung 4](#_Toc133930314)

[5.2 Hướng phát triển 4](#_Toc133930315)

[TÀI LIỆU THAM KHẢO 1](#_Toc133930316)

[PHỤ LỤC 2](#_Toc133930317)

LỜI CAM ĐOAN

Tôi xin cam đoan rằng bài báo cáo này là thành quả của công việc nghiêm túc và không sao chép từ bất kỳ nguồn nào mà không được trích dẫn.

Tất cả thông tin và dữ liệu được thu thập từ các nguồn đáng tin cậy và đã được trích dẫn phù hợp. Bài báo cáo này chính xác và được hoàn thành đúng thời hạn. Tôi cảm ơn các cố vấn và giáo viên hướng dẫn đã hỗ trợ tôi trong quá trình nghiên cứu và viết báo cáo này.

DANH MỤC CÁC HÌNH ẢNH

[Hình 2‑1 Cấu trúc thành phần cuả Hadoop 7](#_Toc133920090)

[Hình 2‑2 Tổng quan môt Hadoop cluster 8](#_Toc133920091)

[Hình 2‑3 Kiến trúc HDFS 12](#_Toc133920092)

[Hình 2‑4 Quá trình đọc file trên HDFS 14](#_Toc133920093)

[Hình 2‑5 Quá trình tạo và ghi dữ liệu lên file trên HDFS 15](#_Toc133920094)

[Hình 2‑6 Cấu trúc topology mạng 19](#_Toc133920095)

[Hình 2‑7 Mô hình Mapreduce của google 25](#_Toc133920096)

[Hình 2‑8 Hàm map 26](#_Toc133920097)

[Hình 2‑9 Hàm reduce 26](#_Toc133920098)

[Hình 2‑10 Kiến trúc thành phần 28](#_Toc133920099)

[Hình 2‑11Cơ chế hoạt động của Hadoop Mapreduce 29](#_Toc133920100)

[Hình 2‑12 Sự liên lạc đầu tiên giữa TaskTracker thực thi Maptask và JobTracker 30](#_Toc133920101)

[Hình 2‑13 Cơ chế hoạt động của MapTask 31](#_Toc133920102)

[Hình 2‑14 Tasktracker hoàn thành Maptask 32](#_Toc133920103)

[Hình 2‑15 Cơ chế hoạt động của ReduceTask 33](#_Toc133920104)

[Hình 2‑16 Task tracker hoàn thành Reduce Task 34](#_Toc133920105)

[Hình 2‑17 Data Locality 35](#_Toc133920106)

[Hình 2‑18 Phát triển ứng dụng Mapreduce trên Hadoop 36](#_Toc133920107)

[Hình 3‑1 Mô tả bài toán phân cụm với 3 cluster 39](#_Toc133920108)

[Hình 3‑2 Phân vùng lãnh hải của mỗi đảo, các vùng khác nhau có màu sắc khác nhau 40](#_Toc133920109)

[Hình 3‑3 Lưu đồ thuật toán Kmean 43](#_Toc133920110)

[Hình 3‑4 Minh họa quá trình phân cụm 44](#_Toc133920111)

[Hình 3‑5 Minh họa cách đánh giá 46](#_Toc133920112)

[Hình 3‑6 Lưu đồ Mapreduce thuật toán Kmean 48](#_Toc133920113)

[Hình 4‑1 Tập dữ liệu khách hàng chưa qua xử lý 51](#_Toc133920114)

[Hình 4‑2 Tập dữ liệu sau khi loại bỏ thuộc tính dư thừa và chuẩn hóa 53](#_Toc133920115)

[Hình 4‑3 Phiên bản java 54](#_Toc133920116)

[Hình 4‑4 Cấu hình file core-site.xml 54](#_Toc133920117)

[Hình 4‑5 cấu hình file Mapred-site.xml 55](#_Toc133920118)

[Hình 4‑6 Cấu hình file hdfs-site.xml 55](#_Toc133920119)

[Hình 4‑7 cấu hình file yarn-site.xml 56](#_Toc133920120)

[Hình 4‑8 Kết quả mô hình với k=3 57](#_Toc133920121)

[Hình 4‑9 Kết quả mô hình với k=4 57](#_Toc133920122)

[Hình 4‑10 Kết quả mô hình với k=5 57](#_Toc133920123)

[Hình 4‑11 Kết quả mô hình với k=6 58](#_Toc133920124)

[Hình 4‑12 Kết quả mô hình với k=7 58](#_Toc133920125)

[Hình 4‑13Kết quả mô hình với k=8 58](#_Toc133920126)

[Hình 4‑14 Kết quả mô hình với k=9 58](#_Toc133920127)

[Hình 4‑15Kết quả mô hình với k=10 58](#_Toc133920128)

**NỘI DUNG TRÌNH BÀY**

Chương 1: Giới thiệu đề tài

## Giới thiệu

Tên đề tài: “Ứng dụng thuật toán Kmean và mô hình mapreduce để phân cụm khách hàng”

Sơ lược: Trong đề tài này, nhóm sẽ thực hiện tìm hiểu kỹ càng nền tảng phát triển

các ứng dụng MapReduce trên framework Hadoop để phát triển các ứng dụng phân

tán. Sau đó, em sẽ tiến hành lập trình và xây dựng thuật toán Kmean dựa trên mô hình Mapreduce để phân cụm tập dữ liệu khách hàng đã thu thập và tiền xử lý trước đó, cuối cùng sẽ đánh giá mô hình và tác dụng của việc ứng dụng Mapreduce vào trong bài toán thực tế.

## Ngữ cảnh và lý do thực hiện đề tài

### Sự bùng phát dữ liệu và bài toán kinh doanh từ dữ liệu khách hàng

Trong thời đại kỹ thuật số hiện nay, dữ liệu được tạo ra và tích lũy với tốc độ chóng mặt. Từ những cuộc gọi điện thoại, email, thông tin đăng ký trên website, cho đến những bản ghi từ cảm biến IoT, dữ liệu khách hàng ngày càng tăng lên với tốc độ không ngừng nghỉ. Điều này đã mở ra một cánh cửa mới cho các doanh nghiệp để khai thác giá trị tiềm năng của dữ liệu khách hàng. [1]

Sự bùng phát dữ liệu khách hàng cung cấp cho các doanh nghiệp rất nhiều thông tin quan trọng về khách hàng của họ, từ sở thích, xu hướng mua sắm, đến quá trình đánh giá sản phẩm và dịch vụ của công ty. Sử dụng các công cụ phân tích dữ liệu, các doanh nghiệp có thể hiểu được hành vi và nhu cầu của khách hàng, từ đó đưa ra các quyết định kinh doanh thông minh và hiệu quả. Ngoài ra, việc thu thập và phân tích dữ liệu khách hàng cũng giúp xây dựng mối quan hệ tốt hơn với khách hàng, đồng thời tạo ra trải nghiệm khách hàng tốt hơn.

Tuy nhiên, việc sử dụng dữ liệu khách hàng cũng đặt ra một số thách thức cho các doanh nghiệp. Trong đó, vấn đề bảo mật dữ liệu là một trong những yếu tố quan trọng nhất. Để giữ cho dữ liệu khách hàng được an toàn và bảo vệ, các doanh nghiệp phải tuân thủ các quy định pháp luật về bảo mật thông tin cá nhân, đồng thời cập nhật các công nghệ bảo mật mới nhất để ngăn chặn các cuộc tấn công mạng và lừa đảo.

Ngoài ra, việc thu thập và phân tích dữ liệu khách hàng cũng đòi hỏi các doanh nghiệp phải có kỹ năng và công cụ phân tích dữ liệu cao cấp. Các doanh nghiệp cần đầu tư vào các công nghệ và nhân sự để có thể hiểu rõ hơn về hành vi của khách hàng của họ, từ đó đưa ra các chiến lược tiếp thị và quản lý khách hàng hiệu quả.

Trong kinh doanh hiện đại, sự bùng phát dữ liệu khách hàng đã trở thành một yếu tố quan trọng để các doanh nghiệp có thể tạo ra giá trị và cạnh tranh trong thị trường. Tuy nhiên, để khai thác được giá trị tiềm năng của dữ liệu khách hàng, các doanh nghiệp cần phải đảm bảo an toàn thông tin cho khách hàng, đồng thời có kỹ năng và công cụ phân tích dữ liệu cao cấp để hiểu rõ hơn về hành vi của khách hàng và đưa ra những quyết định kinh doanh thông minh và hiệu quả.

### Sự ra đời của mô hình Mapreduce

[2]Năm 2004, Google công bố nền tảng MapReduce (thực ra có thể coi MapReduce

là một mô hình lập trình, hay một thuật giải). MapReduce là giải pháp được các kỹ sư

của Google tìm ra khi họ đang cố gắng mở rộng bộ máy tìm kiếm của mình. Có thể

hiểu một cách đơn giản, MapReduce chia việc xử lý thành nhiều khối công việc nhỏ,

phân tán khắp các nút tính toán (tiêu biểu là các server thông thường), rồi thu thập các

kết quả.

Sau khi ra đời, MapReduce nhanh chóng trở thành một đối tượng nghiên cứu và áp

dụng của các doanh nghiệp cần xử lý khối lượng dữ liệu lớn với hai lý do sau:

* MapReduce có thể chạy trên các phần cứng thông thường (commodity

hardware), không đòi hỏi các server chạy MapReduce phải là các máy tính có khả năng

tính toán, lưu trữ và truy xuất mạnh mẽ. Do vậy, chi phí triển khai MapReduce sẽ rẻ

hơn.

* Thứ hai, MapReduce làm đơn giản hoá các giải thuật tính toán phân tán. Với

MapReduce, bạn chỉ cần cung cấp hai hàm Map và Reduce cùng với một số thành phần

xử lý dữ liệu đầu vào. Do vậy, các nhà phát triển ứng dụng phân tán có thể tập trung

nhiều hơn cho phần logic của ứng dụng, bỏ qua các chi tiết phức tạp của việc phân tán

xử lý.

Trước MapReduce, các doanh nghiệp muốn xử lý hàng petabyte (triệu gigabyte)

dữ liệu để tìm mối quan hệ liên quan đến nghiệp vụ phải rất cân nhắc khi đầu tư cho

việc đầy mạo hiểm này vì chi phí và thời gian cần thiết là trở ngại. Sự ra đời của

MapReduce đã mở ra cho các doanh nghiệp cơ hội xử lý các nguồn dữ liệu đồ sộ với

chi phí thấp và thời gian nhanh hơn. Với việc áp dụng MapReduce, Amazon có thể xử

lý được các file log phát sinh trong quá trình bán hàng trên mạng, phục vụ cho việc dự

đoán xu hướng mua hàng của khách hàng, các sản phẩm đang được mua nhiều…

Facebook có thể xử lý được khối lượng hơn 10 tỷ hình ảnh mà họ đang lưu trữ để rút

trích các thông tin về kích thước hình ảnh, phát hiện các hình ảnh xấu.

Cho đến nay, ngoài Google, đã có rất nhiều giải pháp cài đặt bằng nhiều ngôn ngữ khác nhau MapReduce như Qizmt (C#), Skynet (Ruby) và Greenplum (Python, Perl, SQL). Vào cuối đầu năm 2005, Dough Cutting đã áp dụng thành công MapReduce vào ứng dụng Search Engine nguồn mở của mình. Sau đó, nhận ra được các tiềm năng to lớn của MapReduce, Cutting đã tách MapReduce ra thành một dự án riêng biệt với tên gọi Apache Hadoop. Cho đến nay, Hadoop đã trở thành giải pháp nguồn mở hàng đầu hỗ trợ mô hình MapReduce. Hadoop viết bằng Java, tuy nhiên hỗ trợ phát triển

MapReduce trên nhiều ngôn ngữ khác ngoài Java như C++, Pearl, Python.

Sự bùng nổ dữ liệu không gì ngăn được là một thực tế. Khi có các giải pháp sử

dụng MapReduce, chúng ta sẽ có thể nhìn thấy ý nghĩa của petabyte. Năm 2009

MapReduce đã được bầu chọn vào vị trí số một trên danh sách Top 10 công nghệ có

ảnh hướng nhất cùng với các công nghệ

### Lý do thực hiện

Qua các vấn đề đã nêu ra ở trên, ta thấy được những ích lợi sẽ có nếu chúng ta

nắm vững được cách thức phát triển mô hình MapReduce cho bài toán xử lý dữ liệu

lớn. Vì vậy nhóm đã quyết định thực hiện đề tài này với mong muốn tạo ra một cơ sở

lý thuyết và các hướng dẫn kỹ thuật để về vấn đề phát triển ứng dụng phân tán theo mô

hình MapReduce trên một framework nguồn mở: Hadoop. Bên cạnh đó, để minh họa

tốt hơn cho đề tài, nhóm đã chọn Kmean, một thuật toán học máy không giám sát sử

dụng HDFS và MapReduce của Hadoop, để triển khai thực tế.

## Nội dung báo cáo BÀI TẬP LỚN

Bài báo cáo gồm 5 chương chính:

Chương 1: Giới thiệu. Giới thiệu đề tài thực hiện

Chương 2: Giới thiệu nền tảng tính toán phân tán với Hadoop. Trong chương này, nhóm sẽ giới thiệu sơ lược về dự án Hadoop của Apache Software Foundation. Sau đó là nhóm sẽ đi sâu vào hai phần trọng tâm là HDFS và MapReduce Engine. Giới thiệu kiến trúc, sức mạnh, và cách phát triển các ứng dụng phân tán trên Hadoop.

Chương 3: Thuật toán phân cụm Kmean kết hợp mô hình Mapreduce. Trong chương này nhóm sẽ giới thiệu thuật toán Kmean nói riêng và Kmean khi kết hợp với mô hình lập trình phân tán Mapreduce

Chương 4: Báo cáo kết quả thực nghiệm, Trong chương này nhóm sẽ trình bày các thực nghiệm dựa trên tập dữ liệu đã trải qua quá trình tiền xử lý và kết quả thực nghiệm. Thêm vào đó là các đánh giá và kết luận về kết quả này.

Chương 5: Tổng kết. Chương này sẽ nêu lên các kết quả đạt được của đề tài và hướng phát triển.

# NỀN TẢNG TÍNH TOÁN PHÂN TÁN VỚI HADOOP

## 2.1 Giới thiệu Framework Hadoop

### 2.1.1Hadoop là gì

[3]Apache Hadoop định nghĩa: “Apache Hadoop là một framework dùng để chạy những ứng dụng trên 1 cluster lớn được xây dựng trên những phần cứng thông thường 1 . Hadoop hiện thực mô hình Map/Reduce, đây là mô hình mà ứng dụng sẽ được chia nhỏ ra thành nhiều phân đoạn khác nhau, và các phần này sẽ được chạy song song trên nhiều node khác nhau. Thêm vào đó, Hadoop cung cấp 1 hệ thống file phân tán (HDFS) cho phép lưu trữ dữ liệu lên trên nhiều node. Cả Map/Reduce và HDFS đều được thiết kế sao cho framework sẽ tự động quản lý được các lỗi, các hư hỏng về phần cứng của các node.”

[4]Wikipedia định nghĩa:

“Hadoop là một framework nguồn mở viết bằng Java cho phép phát triển các ứng dụng phân tán có cường độ dữ liệu lớn một cách miễn phí. Nó cho phép các ứng dụng có thể làm việc với hàng ngàn node khác nhau và hàng petabyte dữ liệu. Hadoop lấy được phát triển dựa trên ý tưởng từ các công bố của Google về mô hình MapReduce và hệ thống file phân tán Google File System (GFS).”

Vậy ta có thể kết luận như sau:

1. Hadoop là một framework cho phép phát triển các ứng dụng phân tán.

2) 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, Pearl.

3) Hadoop cung cấp một phương tiện lưu trữ dữ liệu phân tán trên nhiều node, hỗ trợ tối ưu hoá lưu lượng mạng, đó là HDFS. HDSF che giấu tất cả các thành phần phân tán, các nhà phát triển ứng dụng phân tán sẽ chỉ nhìn thấy HDFS như một hệ thống file cục bộ bình thường.

4) Hadoop giúp các nhà phát triển ứng dụng phân tán tập trung tối đa vào phần logic của ứng dụng, bỏ qua được một số phần chi tiết kỹ thuật phân tán bên dưới (phần này do Hadoop tự động quản lý).

5) Hadoop là Linux-based. Tức Hadoop chỉ chạy trên môi trường Linux .

### 2.1.2 Lịch sử Hadoop

Hadoop được tạo ra bởi Dough Cutting, người sáng tạo ra Apache Lucene – bộ thư viện tạo chỉ mục tìm kiếm trên text được sử dụng rộng rãi. Hadoop bắt nguồn từ Nutch, một ứng dụng search engine nguồn mở.

Nutch được khởi xướng từ năm 2002, và một hệ thống search engine (gồm crawler và tìm kiếm) nhanh chóng ra đời. Tuy nhiên, các nhà kiến trúc sư của Nutch nhanh chóng nhận ra rằng Nutch sẽ không thể mở rộng ra để có thể thực hiện vai trò searcher engine của mình trên tập dữ liệu hàng tỷ trang web (lúc khả năng của Nutch chỉ có thể crawl tối đa 100 triệu trang). Nguyên nhân chính của giới hạn này là do Nutch lúc này chỉ chạy trên một máy đơn (stand alone) nên gặp phải các khuyết điểm:

* Khả năng lưu trữ bị giới hạn: giả sử mỗi trang web cần 10kb đĩa cứng để lưu, thì với hơn 100 triệu trang ta cần 1 Tetabyte đĩa cứng, và với khối lượng hàng tỷ trang web đang có trên mạng thì cần có tới hàng chục petabye để lưu trữ.
* Tốc độ truy xuất chậm: với khối lượngdữ liệu lớn như vậy, việc truy xuất tuần tự để phân tích dữ liệu và index trở nên rất chậm chạp, và thời gian để đáp ứng các câu truy vấn tìm kiếm (search query) là không hợp lý. Việc phải truy xuất vào các file có kích thước lớn được tạo ra trong quá trình crawl và index cũng là một thách thức lớn.

Năm 2003, Google công bố kiến trúc của hệ thống file phân tán GFS (viết tắt từ Google File System) của họ. Các nhà kiến trúc sư của Nutch thấy rằng GFS sẽ giải quyết được nhu cầu lưu trữ các file rất lớn từ quá trình crawl và index. Năm 2004, họ bắt tay vào việc ứng dụng kiến trúc của GFS vào cài đặt một hệ thống file phân tán nguồn mở có tên Nutch Distributed File System (NDFS).

Năm 2004, Google lại công bố bài báo giới thiệu MapReduce. Vào đầu năm 2005, các nhà phát triển Nutch đã xây dựng được phiên bản MapReduce trên Nutch, và vào giữa năm 2005, tất cả các thuật toán chính của Nutch đều được cải tiến lại để chạy trên nền NDFS và MapReduce.

NDFS và MapRecude trong Nutch đã nhanh chóng tìm được các ứng dụng của mình bên ngoài lĩnh vực search engine, và vào tháng hai 2006 Dough Cutting đã tách riêng NDFS và MapReduce ra để hình thành một dự án độc lập có tên Hadoop. Cùng thời gian này, Dough Cutting gia nhập vào Yahoo!. Tại đây ông được tạo một môi trường tuyệt vời để phát triển Hadoop và vào tháng 2 năm 2008 Yahoo đã công bố sản phẩm search engine của họ được xây dựng trên một Hadoop cluster có kích thước 10.000 nhân vi xử lý.

Năm 2008, Apache đã đưa Hadoop lên thành dự án ở top-level Apache Software Foundation, nhằm xác nhận sự thành công và các áp dụng rộng rãi của Hadoop. Vào thời gian này, Hadoop được sử dụng bởi rất nhiều công ty ngoài Yahoo! như Last.fm, Facebook, New York Times.

Năm 2008, Hadoop đã phá kỷ lục thế giới về sắp xếp một terabyte dữ liệu. Chạy trên một cluster gồm 910 node, Hadoop đã sắp xếp một terabyte dữ liệu trong vòng 209 giây, phá kỷ lục cũ là 297 giây. Sau đó ít lâu, Google công bố ứng dụng chạy trên MapReduce của họ đã sắp xếp được một terabyte dữ liệu trong 68 giây. Vào tháng 5 năm 2009, một đội các nhà phát triển của Yahoo! đã dùng Hadoop để sắp xếp một terabyte dữ liệu trong vòng 62 giây.

### 2.1.3 Các thành phần của Hadoop

Ngày nay, ngoài NDFS (đã được đổi tên lại thành HDFS – Hadoop Distributed File System) và MapReduce, đội ngũ phát triển Hadoop đã phát triển các dự án con dựa trên HDFS và MapReduce. Hiện nay, Hadoop gồm có các dự án con sau:

A close-up of a colorful label

Description automatically generated with low confidence

Hình ‑ Cấu trúc thành phần cuả Hadoop

* HDFS: hệ thống file phân tán, cung cấp khả năng lưu trữ dữ liệu khổng lồ và tính năng tối ưu hoá việc sử dụng băng thông giữa các node. HDFS có thể được sử dụng để chạy trên một cluster lớn với hàng chục ngàn node.
* HBase: một cơ sở dữ liệu phân tán, theo hướng cột (colunm-oriented). HBase sử dụng HDFS làm hạ tầng cho việc lưu trữ dữ liệu bên dưới, và cung cấp khả năng tính toán song song dựa trên MapReduce.
* Hive: một data warehouse phân tán. Hive quản lý dữ liệu được lưu trữ trên HDFS và cung cấp một ngôn ngữ truy vấn dựa trên SQL.
* Chukwa: một hệ thống tập hợp và phân tích dữ liệu. Chukwa chạy các collector (các chương trình tập hợp dữ liệu), các collector này lưu trữ dữ liệu trên HDFS và sử dụng MapReduce để phát sinh các báo cáo.
* Pig: ngôn ngữ luồng dữ liệu cấp cao và framework thực thi dùng cho tính toán song song. Trong khuôn khổ của luận văn này, chúng tôi chỉ nghiên cứu hai phần quan trọng nhất của Hadoop, đó là HDFS và MapReduce.

### 2.1.4 Ứng dụng của Hadoop trong một số công ty

Ngày nay, ngoài Yahoo!, có nhiều công ty sử dụng Hadoop như là một công cụ để lưu trữ và phân tích dữ liệu trên các khối dữ liệu lớn như:

* Twitter: sử dụng Hadoop để xử lý tweets (các bài viết văn bản lên đến 140 ký tự hiển thị trên profile của tác giả), logs và các nguồn dữ liệu phát sinh trong quá trình hoạt động của Twitter.
* Facebook: Sử dụng Hadoop để lưu trữ các log nội bộ và kích thước của nguồn dữ liệu. Các dữ liệu này được dùng làm nguồn cho các báo cáo phân tích và máy học. 14 Hiện tại, facebook có 2 Hadoop cluster chính: một cluster 1100 máy với 8800 nhân và 12 Petabyte ổ cứng lưu trữ.
* A9.com – Amazon: Sử dụng Hadoop để đánh giá chỉ số tìm kiếm sản phẩm trên Amazon, xử lý đến hàng triệu Session mỗi ngày. Các cluster của A9.com có độ lớn từ 1-100 node.

Và còn rất nhiều công ty hiện đang sử dụng Hadoop vào việc lưu trữ và xử lý dữ liệu, đặc biệt cho các nguồn dữ liệu lớn với kích thước lên tới hàng petabyte.

### 2.1.5 Tổng quan của một Hadoop cluster

[5]Như đã giới thiệu ở các phần trên, HDFS và MapReduce là hai thành phần chính của một Hadoop cluster. Nhìn chung, kiến trúc của Hadoop là kiến trúc master-slave, và cả hai thành phần HDFS và MapReduce đều tuân theo kiến trúc master-slave này. Kiến trúc một Hadoop cluster như sau:

A screenshot of a computer program

Description automatically generated with low confidence

Hình ‑ Tổng quan môt Hadoop cluster

Trên một hadoop cluster, có duy nhất một node chạy NameNode, một node chạy JobTracker (NameNode và JobTracker có thể nằm trên cùng một máy vật lý, tuy nhiên 16 trên các cluster thật sự với hàng trăm, hàng nghìn node thì thường phải tách riêng NameNode và JobTracker ra các máy vật lý khác nhau). Có nhiều node slave, mỗi node slave thường đóng 2 vai trò: một là DataNode, hai là TaskTracker.

NameNode và DataNode chịu trách nhiệm vận hành hệ thống file phân tán HDFS với vai trò cụ thể được phân chia như sau:

* NameNode: đóng vai trò là master của hệ thống HDFS, quản lý các meta-data của hệ thống HDFS như file system space, danh sách các file trên hệ thống và các block id tương ứng của từng file, quản danh sách slave và tình trạng hoạt động của các DataNode (live hay dead) thông qua các hearbeat 3 , điều hướng quá trình đọc/ghi dữ liệu từ client lên các DataNode.
* DataNode: chứa các block dữ liệu thực sự của các file trên HDFS, chịu trách nhiệm đáp ứng các yêu cầu đọc/ghi dữ liệu từ client, đáp ứng các yêu cầu tạo/xoá các block dữ liệu từ NameNode.

JobTracker và TaskTracker chịu trách nhiệm duy trì bộ máy MapReduce, nhận và thực thi các MapReduce Job 4 . Vai trò cụ thể như sau:

* JobTracker: tiếp nhận các yêu cầu thực thi các MapReduce job, phân chia job này thành các task và phân công cho các TaskTracker thực hiện, quản lý tình trạng thực hiện các task của TaskTracker và phân công lại nếu cần. JobTracker cũng quản lý danh sách các node TaskTracker và tình trạng của từng node thông qua hearbeat.
* TaskTracker: nhận các task từ JobTracker và thực hiện task.

Ngoài ra trên một Hadoop cluster còn có SecondaryNameNode: duy trì một bản sao của meta-data trên NameNode và bản sao này sẽ được dùng để phục hồi lại NameNode nếu NameNode bị hư hỏng.

## 2.2 Hadoop Distributed File System (HDFS)

Khi kích thước của tập dữ liệu vượt quá khả năng lưu trữ của một máy tính, tất yếu sẽ dẫn đến nhu cầu phân chia dữ liệu lên trên nhiều máy tính. Các hệ thống tập tin quản lý việc lưu trữ dữ liệu trên một mạng nhiều máy tính gọi là hệ thống tập tin phân tán. Do hoạt động trên môi trường liên mạng, nên các hệ thống tập tin phân tán phức tạp hơn rất nhiều so với một hệ thống file cục bộ. Ví dụ như một hệ thống file phân tán phải quản lý được tình trạng hoạt động (live/dead) của các server tham gia vào hệ thống file.

Hadoop mang đến cho chúng ta hệ thống tập tin phân tán HDFS (viết tắt từ Hadoop Distributed File System) với nỗ lực tạo ra một nền tảng lưu trữ dữ liệu đáp ứng cho một khối lượng dữ liệu lớn và chi phí rẻ. Trong chương này chúng tôi sẽ giới thiệu kiến trúc của HDFS cũng như các sức mạnh của nó.

### 2.2.1 Giới thiệu

HDFS ra đời trên nhu cầu lưu trữ dữ liệu của Nutch, một dự án Search Engine nguồn mở. HDFS kế thừa các mục tiêu chung của các hệ thống file phân tán trước đó như độ tin cậy, khả năng mở rộng và hiệu suất hoạt động. Tuy nhiên, HDFS ra đời trên nhu cầu lưu trữ dữ liệu của Nutch, một dự án Search Engine nguồn mở, và phát triển để đáp ứng các đòi hỏi về lưu trữ và xử lý của các hệ thống xử lý dữ liệu lớn với các đặc thù riêng. Do đó, các nhà phát triển HDFS đã xem xét lại các kiến trúc phân tán trước đây và nhận ra các sự khác biệt trong mục tiêu của HDFS so với các hệ thống file phân tán truyền thống.

Thứ nhất, các lỗi về phần cứng sẽ thường xuyên xảy ra. Hệ thống HDFS sẽ chạy trên các cluster với hàng trăm hoặc thậm chí hàng nghìn node. Các node này được xây 18 dựng nên từ các phần cứng thông thường, giá rẻ, tỷ lệ lỗi cao. Chất lượng và số lượng của các thành phần phần cứng như vậy sẽ tất yếu dẫn đến tỷ lệ xảy ra lỗi trên cluster sẽ cao. Các vấn đề có thể điểm qua như lỗi của ứng dụng, lỗi của hệ điều hành, lỗi đĩa cứng, bộ nhớ, lỗi của các thiết bị kết nối, lỗi mạng, và lỗi về nguồn điện… Vì thế, khả năng phát hiện lỗi, chống chịu lỗi và tự động phục hồi phải được tích hợp vào trong hệ thống HDFS.

Thứ hai, kích thước file sẽ lớn hơn so với các chuẩn truyền thống, các file có kích thước hàng GB sẽ trở nên phổ biến. Khi làm việc trên các tập dữ liệu với kích thước nhiều TB, ít khi nào người ta lại chọn việc quản lý hàng tỷ file có kích thước hàng KB, thậm chí nếu hệ thống có thể hỗ trợ. Điều chúng muốn nói ở đây là việc phân chia tập dữ liệu thành một số lượng ít file có kích thước lớn sẽ là tối ưu hơn. Hai tác dụng to lớn của điều này có thể thấy là giảm thời gian truy xuất dữ liệu và đơn giản hoá việc quản lý các tập tin.

Thứ ba, hầu hết các file đều được thay đổi bằng cách append dữ liệu vào cuối file hơn là ghi đè lên dữ liệu hiện có. Việc ghi dữ liệu lên một vị trí ngẫu nhiên trong file không hề tồn tại. Một khi đã được tạo ra, các file sẽ trở thành file chỉ đọc (read-only), và thường được đọc một cách tuần tự. Có rất nhiều loại dữ liệu phù hợp với các đặc điểm trên. Đó có thể là các kho dữ liệu lớn để các chương trình xử lý quét qua và phân tích dữ liệu. Đó có thể là các dòng dữ liệu được tạo ra một cách liên tục qua quá trình chạy các ứng dụng (ví dụ như các file log). Đó có thể là kết quả trung gian của một máy này và lại được dùng làm đầu vào xử lý trên một máy khác. Và do vậy, việc append dữ liệu vào file sẽ trở thành điểm chính để tối ưu hoá hiệu suất.

Đã có rất nhiều Hadoop cluster chạy HDFS trên thế giới. Trong đó nổi bật nhất là của Yahoo với một cluster lên đến 1100 node với dung lượng HDFS 12 PB. Các công ty khác như Facebook, Adode, Amazon cũng đã xây dựng các cluster chạy HDFS với dung lượng hàng trăm, hàng nghìn TB.

### 2.2.2 Tổng quan thiết kế của HDFS

#### 2.2.2.1 Một số giả định

Để tạo ra một hệ thống file phù hợp với nhu cầu sử dụng, các nhà thiết kế HDFS đã khảo sát thực tế hệ thống và đưa ra các giả định sau về hệ thống:

* Hệ thống được xây dựng trên các phần cứng giá rẻ với khả năng hỏng hóc cao. Do dó HDFS phải tự động phát hiện, khắc phục, và phục hồi kịp lúc khi các thành phần phần cứng bị hư hỏng.
* Hệ thống sẽ lưu trữ một số lượng khiêm tốn các tập tin có kích thước lớn. Các nhà thiết kế giả định rằng sẽ có một vài triệu file, với kích thước mỗi file khoảng vài trăm MB hoặc lớn hơn. Các file có kích thước nhiều GB sẽ phổ biến và cần được quản lý hiệu quả. Các file kích thước nhỏ cũng được hỗ trợ, tuy nhiên không cần tối ưu hoá các trường hợp này.
* HDFS không phải là một hệ thống file dành cho các mục đích chung. HDFS được thiết kế dành cho các ứng dụng dạng xử lý khối (batch processing). Do đó, các file trên HDFS một khi được tạo ra, ghi dữ liệu và đóng lại thì không thể bị chỉnh sữa được nữa. Điều này làm đơn giản hoá đảm bảo tính nhất quán của dữ liệu và cho phép truy cập dữ liệu với thông lượng cao.

2.2.2.2 Kiến trúc HDFS

Giống như các hệ thống file khác, HDFS duy trì một cấu trúc cây phân cấp các file, thư mục mà các file sẽ đóng vai trò là các node lá. Trong HDFS, mỗi file sẽ được chia ra làm một hay nhiều block và mỗi block này sẽ có một block ID để nhận diện. Các block của cùng một file (trừ block cuối cùng) sẽ có cùng kích thước và kích thước này được gọi là block size của file đó. Mỗi block của file sẽ được lưu trữ thành ra nhiều bản sao (replica) khác nhau vì mục đích an toàn dữ liệu (xem phần 2.2.4.2 )

HDFS có một kiến trúc master/slave. Trên một cluster chạy HDFS, có hai loại node là Namenode và Datanode. Một cluster có duy nhất một Namenode và có một hay nhiều Datanode.

Namenode đóng vai trò là master, chịu trách nhiệm duy trì thông tin về cấu trúc cây phân cấp các file, thư mục của hệ thống file và các metadata khác của hệ thống file. Cụ thể, các Metadata mà Namenode lưu trữ gồm có:

* File System Namespace: là hình ảnh cây thư mục của hệ thống file tại một thời điểm nào đó. File System namespace thể hiện tất các các file, thư mục có trên hệ thống file và quan hệ giữa chúng.
* Thông tin để ánh xạ từ tên file ra thành danh sách các block: với mỗi file, ta có một danh sách có thứ tự các block của file đó, mỗi Block đại diện bởi Block ID.
* Nơi lưu trữ các block: các block được đại diện một Block ID. Với mỗi block ta có một danh sách các DataNode lưu trữ các bản sao của block đó.

Các Datanode sẽ chịu trách nhiệm lưu trữ các block thật sự của từng file của hệ thống file phân tán lên hệ thống file cục bộ của nó. Mỗi 1 block sẽ được lưu trữ như là 1 file riêng biệt trên hệ thống file cục bộ của DataNode.

Kiến trúc của HDFS được thể hiện qua sơ đồ dưới đây :

A picture containing text, diagram, plan, technical drawing

Description automatically generated

Hình ‑ Kiến trúc HDFS

Namenode sẽ chịu trách nhiệm điều phối các thao tác truy cập (đọc/ghi dữ liệu) của client lên hệ thống HDFS. Và tất nhiên, do các Datanode là nơi thật sự lưu trữ các block của các file trên HDFS, nên chúng sẽ là nơi trực tiếp đáp ứng các thao tác truy cập này. Chẳng hạn như khi client của hệ thống muốn đọc 1 file trên hệ thống HDFS, client này sẽ thực hiện một request (thông qua RPC) đến Namenode để lấy các metadata của file cần đọc. Từ metadata này nó sẽ biết được danh sách các block của file và vị trí của các Datanode chứa các bản sao của từng block. Client sẽ truy cập vào 22 các Datanode để thực hiện các request đọc các block. Chi tiết về các quá trình đọc/ghi dữ liệu của client lên trên HDFS sẽ được giới thiệu kỹ hơn ở phần 2.2.2.3.1 và 2.2.2.3 .

Namenode thực hiện nhiệm vụ của nó thông qua một daemon tên namenode chạy trên port 8021. Mỗi Datanode server sẽ chạy một daemon datanode trên port 8022.

Định kỳ, mỗi DataNode sẽ báo cáo cho NameNode biết về danh sách tất cả các block mà nó đang lưu trữ, Namenode sẽ dựa vào những thông tin này để cập nhật lại các metadata trong nó. Cứ sau mỗi lần cập nhật lại như vậy, metadata trên namenode sẽ đạt được tình trạng thống nhất với dữ liệu trên các Datanode. Toàn bộ trạng thái của metadata khi đang ở tình trạng thống nhất này được gọi là một checkpoint. Metadata ở trạng thái checkpoint sẽ được dùng để nhân bản metadata dùng cho mục đích phục hồi lại NameNode nếu NameNode bị lỗi (xem phần 2.2.4.3 ).

#### 2.2.2.3 NameNode và quá trình tương tác giữa client và HDFS

Việc tồn tại duy nhất một NameNode trên một hệ thống HDFS đã làm đơn giản hoá thiết kế của hệ thống và cho phép NameNode ra những quyết định thông minh trong việc sắp xếp các block dữ liệu lên trên các DataNode dựa vào các kiến thức về môi trường hệ thống như: cấu trúc mạng, băng thông mạng, khả năng của các DataNode… Tuy nhiên, cần phải tối thiểu hoá sự tham gia của NameNode vào các quá trình đọc/ghi dữ liệu lên hệ thống để tránh tình trạng nút thắt cổ chai (bottle neck). Client sẽ không bao giờ đọc hay ghi dữ liệu lên hệ thống thông qua NameNode. Thay vào đó, client sẽ hỏi NameNode xem nên liên lạc với DataNode nào để truy xuất dữ liệu. Sau đó, client sẽ cache thông tin này lại và kết nối trực tiếp với các DataNode để thực hiện các thao tác truy xuất dữ liệu.

Chúng ta sẽ mổ xẻ quá trình đọc một file từ HDFS và ghi một file lên HDFS thông qua việc tương tác giữa các đối tượng từ phía client lên HDFS.

2.2.2.3.1. Quá trình đọc file

Sơ đồ sau miêu tả rõ quá trình client đọc một file trên HDFS.

A picture containing text, screenshot, line, diagram

Description automatically generated

Hình ‑ Quá trình đọc file trên HDFS

Đầu tiên, client sẽ mở file cần đọc bằng cách gửi yêu cầu đọc file đến NameNode (1).Sau đó NameNode sẽ thực hiện một số kiểm tra xem file được yêu cầu đọc có tồn tại không, hoặc file cần đọc có đang ở trạng thái “khoẻ mạnh” hay không. Nếu mọi thứ đều ổn, NameNode sẽ gửi danh sách các block (đại diện bởi Block ID) của file cùng với địa chỉ các DataNode chứa các bản sao của block này.

Tiếp theo, client sẽ mở các kết nối tới Datanode, thực hiện một RPC để yêu cầu nhận block cần đọc và đóng kết nối với DataNode (3). Lưu ý là với mỗi block ta có thể có nhiều DataNode lưu trữ các bản sao của block đó. Client sẽ chỉ đọc bản sao của block từ DataNode “gần” nhất. Việc tính khoảng cách giữa hai node trên cluster sẽ được trình bày ở phần 2.2.3.1 .

Client sẽ thực hiện việc đọc các block lặp đi lăp lại cho đến khi block cuối cùng của file được đọc xong. Quá trình client đọc dữ liệu từ HDFS sẽ transparent với người dùng hoặc chương trình ứng dụng client, người dùng sẽ dùng một tập API của Hadoop để tương tác với HDFS, các API này che giấu đi quá trình liên lạc với NameNode và kết nối các DataNode để nhận dữ liệu.

Nhận xét: Trong quá trình một client đọc một file trên HDFS, ta thấy client sẽ trực tiếp kết nối với các Datanode để lấy dữ liệu chứ không cần thực hiện gián tiếp qua NameNode (master của hệ thống). Điều này sẽ làm giảm đi rất nhiều việc trao đổi dữ liệu giữa client NameNode, khối lượng luân chuyển dữ liệu sẽ được trải đều ra khắp cluster, tình trạng bottle neck sẽ không xảy ra. Do đó, cluster chạy HDFS có thể đáp ứng đồng thời nhiều client cùng thao tác tại một thời điểm.

2.2.2.3.2. Ghi file

Tiếp theo, ta sẽ khảo sát quá trình client tạo một file trên HDFS và ghi dữ liệu lên file đó. Sơ đồ sau mô tả quá trình tương tác giữa client lên hệ thống HDFS

A picture containing text, screenshot, diagram, line

Description automatically generated

Hình ‑ Quá trình tạo và ghi dữ liệu lên file trên HDFS

Đầu tiên, client sẽ gửi yêu cầu đến NameNode tạo một file entry lên File System Namespace (1). File mới được tạo sẽ rỗng, tức chưa có một block nào. Sau đó, NameNode sẽ quyết định danh sách các DataNode sẽ chứa các bản sao của file cần gì và gửi lại cho client (2)

Client sẽ chia file cần gì ra thành các block, và với mỗi block client sẽ đóng gói thành một packet. Lưu ý là mỗi block sẽ được lưu ra thành nhiều bản sao trên các DataNode khác nhau (tuỳ vào chỉ số độ nhân bản của file).

Client gửi packet cho DataNode thứ nhất, DataNode thứ nhất sau khi nhận được packet sẽ tiến hành lưu lại bản sao thứ nhất của block. Tiếp theo DataNode thứ nhất sẽ gửi packet này cho DataNode thứ hai để lưu ra bản sao thứ hai của block. Tương tự 26 DataNode thứ hai sẽ gửi packet cho DataNode thứ ba. Cứ như vậy, các DataNode cũng lưu các bản sao của một block sẽ hình thành một ống dẫn dữ liệu data pile.

Sau khi DataNode cuối cùng nhận thành được packet, nó sẽ gửi lại cho DataNode thứ hai một gói xác nhận rằng đã lưu thành công (4). Và gói thứ hai lại gửi gói xác nhận tình trạng thành công của hai DataNode về DataNode thứ nhất. Client sẽ nhận được các báo cáo xác nhận từ DataNode thứ nhất cho tình trạng thành công của tất cả DataNode trên data pile.

Nếu có bất kỳ một DataNode nào bị lỗi trong quá trình ghi dữ liệu, client sẽ tiến hành xác nhận lại các DataNode đã lưu thành công bản sao của block và thực hiện một hành vi ghi lại block lên trên DataNode bị lỗi.

Sau khi tất cả các block của file đều đã đươc ghi lên các DataNode, client sẽ thực hiên một thông điệp báo cho NameNode nhằm cập nhật lại danh sách các block của file vừa tạo. Thông tin Mapping từ Block Id sang danh sách các DataNode lưu trữ sẽ được NameNode tự động cập nhật bằng các định kỳ các DataNode sẽ gửi báo cáo cho NameNode danh sách các block mà nó quản lý.

Nhận xét: cũng giống như trong quá trình đọc, client sẽ trực tiếp ghi dữ liệu lên các DataNode mà không cần phải thông qua NameNode. Một đặc điểm nổi trội nữa là khi client ghi một block với chỉ số replication là n, tức nó cần ghi block lên n DataNode, nhờ cơ chế luân chuyển block dữ liệu qua ống dẫn (pipe) nên lưu lượng dữ liệu cần write từ client sẽ giảm đi n lần, phân đều ra các DataNode trên cluter.

#### 2.2.2.4 Block size

Như ta đã biết, trên đĩa cứng, đơn vị lưu trữ dữ liệu nhỏ nhất không phải là byte, bit hay KB mà đó là một block. Block size của đĩa cứng sẽ quy định lượng dữ liệu nhỏ nhất mà ta có thể đọc/ghi lên đĩa. Các file system trên đĩa đơn (phân biệt với các 27 distributed file system) như của Windows hay Unix, cũng sử dụng block như là đơn vị trao đổi dữ liệu nhỏ nhất, block size trên các file system này thường là khoảng nhiều lần block size trên đĩa cứng.

HDFS cũng chia file ra thành các block, và mỗi block này sẽ được lưu trữ trên Datanode thành một file riêng biệt trên hệ thống file local của nó. Đây cũng sẽ là đơn vị trao đổi dữ liệu nhỏ nhất giữa HDFS và client của nó. Block size là một trong những điểm quan trọng trong thiết kế HDFS. Block size mặc định của HDFS là 64MB, nhưng thông thường trên các hệ thống lớn người ta dùng block size là 128 MB, lớn hơn block size của các hệ thống file truyền thống rất nhiều.

Việc sử dụng block size lớn, tức sẽ giảm số lượng block của một file, mang lại một số thuận lợi. Đầu tiên, nó sẽ làm giảm nhu cầu tương tác với NameNode của client vì việc đọc/ghi trên một block chỉ cần một lần tương tác với NameNode để lấy Block ID và nơi lưu block đó.

Thứ hai, với block size lớn, client sẽ phải tương tác với DataNode ít hơn. Mỗi lần client cần đọc một Block Id trên DataNode, client phải tạo một kết nối TCP/IP đến DataNode. Việc giảm số lượng block cần đọc sẽ giảm số lượng kết nối cần tạo, client sẽ thường làm việc với một kết nối bền vững hơn là tạo nhiều kết nối.

Thứ ba, việc giảm số lượng block của một file sẽ làm giảm khối lượng metadata trên NameNode. Điều này giúp chúng ta có thể đưa toàn bộ metadata vào bộ nhớ chính.

Mặt khác, việc sử dụng block size lớn sẽ dẫn đến việc một file nhỏ chỉ có một vài block, thường là chỉ có một. Điều này dẫn đến việc các DataNode lưu block này sẽ trở thành điểm nóng khi có nhiều client cùng truy cập vào file. Tuy nhiên hệ thống HDFS đa phần chỉ làm việc trên các file có kích thước lớn (như đã đề cập ở phần Các Giả Định) với nhiều block nên sự bất tiện này là không đáng kể trong thực tiễn.

#### 2.2.2.5 Metadata

NameNode lưu trữ ba loại metadata chính: file system namespace, thông tin để ánh xạ file thành các block và thông tin nơi lưu trữ (địa chỉ/tên DataNode) của các block. Tất cả các metadata này đều được lưu trữ trong bộ nhớ chính của NameNode. Hai loại metadata đầu tiên còn được lưu trữ bền vững bằng cách ghi nhật ký các thay đổi vào EditLog và FsImage được lưu trữ trên hệ thống file local của NameNode. Việc sử dụng nhật ký để lưu trữ các thay đổi của metadata giúp chúng ta có thể thay đổi trạng thái của metadata một cách thuận tiện hơn (chỉ cần thay metadata trên bộ nhớ và ghi nhật ký xuống EditLog, không cần cập nhật tất cả metadata xuống đĩa cứng). NameNode sẽ không lưu trữ bền vững thông tin về nơi lưu trữ của các block. Thay vào đó, NameNode sẽ hỏi các DataNode về thông tin các block mà nó lưu trữ mỗi khi một DataNode tham gia vào cluster.

2.2.2.5.1. Cấu trúc dữ liệu lưu trong bộ nhớ

Vì cấu trúc dữ liệu được lưu trong bộ nhớ chính, nên các thao tác của NameNode sẽ nhanh. Hơn nữa, điều này sẽ làm cho việc scan toàn bộ metadata diễn ra một cách dễ dàng.

Việc scan định kỳ này được dùng để cài đặt các tính năng như bộ thu nhặt rác (gabage collection), cân bằng khối lượng dữ liệu lưu trữ giữa các DataNode. Một bất lợi của việc lưu trữ toàn bộ metadata trong bộ nhớ là số lượng tổng số lượng block trên cluster có thể bị giới hạn bởi dung lượng bộ nhớ của NameNode. NameNode cần 64 byte để lưu trữ metadata của mỗi block, với một triệu block ta cần gần 64 MB.

Tuy nhiên, nếu cần mở rộng hệ thống HDFS, thì việc mua thêm bộ nhớ mất chi phí không quá cao.

2.2.2.5.2. Vị trí lưu các block

NameNode sẽ không lưu trữ bền vững thông tin về nơi lưu trữ các bản sao của các block. Nó chỉ hỏi các DataNode các thông tin đó lúc DataNode khởi động. NameNode sẽ giữ cho thông tin nơi lưu trữ các block đươc cập nhật vì một điều đơn giản: NameNode điều khiển tất cả các thao tác sắp đặt các bản sao của các block lên các DataNode và quản lý tình trạng các DataNode bằng thông điệp HearBeat.

2.2.2.5.3. Nhật ký thao tác

EditLog chứa tất cả nhật ký các thao tác làm thay đổi tình trạng của metadata. Ví dụ như việc tạo một file mới trong HDFS làm cho NameNode thêm một record mới vào trong EditLog ghi nhận hành động tạo file này. Hoặc việc thay đổi chỉ số độ sao chép (replication level) của một file cũng tạo ra trên EditLog một record. Vì EditLog rất quan trọng nên ta phải lưu ghi dữ liệu một cách tin cậy xuống EditLog và là không cho client thấy được sự thay đổi trên hệ thống cho đến khi sự thay đổi được cập nhật lên metadata và đã ghi nhật ký bền vững xuống EditLog. EditLog được lưu trữ như một file trên hệ thống file cục bộ của NameNode. EditLog sẽ được dùng trong quá trình phục hồi hệ thống với SecondaryNameNode. Điều này sẽ được mô tả chi tiết trong phần 2.2.4.3 .

Toàn bộ File System Namespace và thông tin ánh xạ từ file thành các block sẽ được lưu trữ trong một file tên FsImage trên hệ thống file cục bộ của NameNode.

### 2.2.3 Các tính năng của NameNode

#### 2.2.3.1 Nhận biết cấu trúc topology của mạng

Trong bối cảnh xử lý dữ liệu với kích thước lớn qua môi trường mạng, việc nhận biết ra giới hạn về băng thông giữa các node là một yếu tố quan trọng để Hadoop đưa ra các quyết định trong việc phân bố dữ liệu và phân tán tính toán. Ý tưởng đo băng thông giữa hai node có vẻ như hợp lý, tuy nhiên làm được điều này là khá khó khăn (vì 30 việc đo băng thông mạng cần được thực hiện trong một môi trường “yên tĩnh”, tức tại thời điểm đo thì không được xảy ra việc trao đổi dữ liệu qua mạng). Vì vậy, Hadoop đã sử dụng cấu trúc topology mạng của cluster để định lượng khả năng truyền tải dữ liệu giữa hai node bất kỳ.

Hadoop nhận biết cấu trúc topology mạng của cluster qua một cấu trúc cây phân cấp. Cấu trúc này sẽ giúp Hadoop nhận biết được “khoảng cách” giữa hai node trên cluster. Trên cấu trúc phân cấp này, các bridge sẽ đóng vai trò là các “node trong” để phân chia mạng ra thành các mạng con (subnet). 2 node có cùng một node cha (cùng nằm trên một mạng con) thì được xem như là “nằm trên cùng một rack”.

Hadoop đưa ra một khái niệm là “địa chỉ mạng” để xác định vị trí tương đối của các node. “Địa chỉ mạng” của một node bất kỳ sẽ là đường dẫn từ node gốc đến node xác định. Ví dụ địa chỉ mạng của Node 1 của hình bên dưới sẽ là /Swicth 1/Rack 1/ Node 1.

Hadoop sẽ tính toán “khoảng cách” giữa hai node bất kỳ đơn giản bằng tổng khoảng cách của 2 node đến node cha chung gần nhất. Ví dụ như theo hình bên dưới, khoảng cách giữa Node 1 và Node 2 là 2, khoảng cách giữa Node 1 và Node 4 là 4.

A picture containing design

Description automatically generated with medium confidence

Hình ‑ Cấu trúc topology mạng

Hadoop đưa ra một số giả định sau đây về rack:

* Băng thông giảm dần theo thứ tự sau đây. 1. Các tiến trình trên cùng một node. 2. Các node khác nhau trên cùng một rack. 3. Các node nằm không cùng nằm trên một rack.
* Hai node có “khoảng cách” càng gần nhau thì có băng thông giữa hai node đó càng lớn. Giả định này khẳng định lại giả định đầu tiên.
* Khả năng hai node nằm trên cùng một rack cùng bị lỗi (death) là cao hơn so với hai node nằm trên hai rack khác nhau. Điều này sẽ được ứng dụng khi NameNode thực hiện sắp đặt các bản sao cho một block xuống các DataNode. 32

Ta sẽ thấy rõ tác dụng của các giả định này trong các phần 2.2.3.2 , 2.2.3.3 .

#### 2.2.3.2 Sắp xếp bản sao của các block lên các DataNode

Như ta đã biết, trên HDFS, một file được chia ra thành nhiều block, và mỗi block sẽ được lưu trữ ra thành N bản sao trên N DataNode khác nhau, N được gọi là chỉ số mức độ sao chép (replication level). Với mỗi file trên HDFS, ta có thể quy định một chỉ số replication level khác nhau. Chỉ số này càng cao thì file càng “an toàn”. Do mỗi block của file sẽ được lưu trữ ra càng nhiều bản sao trên các DataNode khác nhau.

Một vấn đề được đặt ra là: “NameNode sẽ chọn những DataNode nào để lưu các bản sao của các một block?”. Ở đây, chúng ta sẽ có một sự cân bằng giữa ba yếu tố: độ tin cậy, băng thông đọc và băng thông ghi dữ liệu. Ví dụ như nếu ta ghi tất các bản sao của một block trên duy nhất một DataNode, thì băng thông ghi sẽ tối ưu vì data pile (xem lại 2.2.2.3.2. , Quá trình ghi file) chỉ xảy ra trên một node duy nhất. Tuy nhiên sự tin cậy sẽ tối thiểu (vì nếu DataNode đó “chết” thì tất cả các bản sao block dữ liệu đó cũng mất hết). Một ví dụ khác, nếu ta lưu các bản sao của một block lên nhiều DataNode thuộc các rack khác nhau. Điều này làm cho block đó an toàn, vì khả năng các node thuộc các rack khác nhau cũng “chết” là khó xảy ra. Tuy nhiên băng thông ghi dữ liệu sẽ thấp vì data pile trải ra trên nhiều node thuộc các rack khác nhau. Vì sự cân bằng này các yếu tố trên chỉ mang tính chất tương đối, nên xuất hiện khá nhiều chiến lược cho việc sắp xếp bản sao của các block lên các DataNode. Từ bản Hadoop 0.17.0 trở đi, chiến lược này đã được cố định và theo nguyên tắt là “phân tán các bản sao của từng block ra khắp cluster”.

Theo chiến lược này, bản sao đầu tiên của một block dữ liệu sẽ được đặt trên cùng node với client (nếu chương trình client ghi dữ liệu cũng thuộc cluster, ngược lại, NameNode sẽ chọn ngẫu nhiên một DataNode). Bản sao thứ hai sẽ được đặt trên một DataNode ngẫu nhiên nằm trên rack khác với node lưu bản sao đầu tiên. Bản sao thứ 33 ba, sẽ được đặt trên một DataNode nằm cùng rack với node lưu bản sao thứ hai. Các bản sao xa hơn được đặt trên các DataNode được chọn ngẫu nhiên.

Nhìn chung, chiến lược này đảm bảo cân bằng được cả ba yếu tố là độ tin cậy (một block sẽ được lưu trên hai rack khác nhau), băng thông ghi (data pile chỉ đi qua hai rack) và băng thông đọc (vì client sẽ có được hai sự lựa chọn xem nên đọc trên rack nào).

#### 2.2.3.3 Cân bằng cluster

Theo thời gian sự phân bố của các block dữ liệu trên các DataNode có thể trở nên mất cân đối, một số node lưu trữ quá nhiều block dữ liệu trong khi một số node khác lại ít hơn. Một cluster bị mất cân bằng có thể ảnh hưởng tới sự tối ưu hoá MapReduce (MapReduce locality, xem phần 2.3.2.2.3. ) và sẽ tạo áp lực lên các DataNode lưu trữ quá nhiều block dữ liệu (lưu lượng truy cập từ client, dung lượng lưu trữ lớn). Vì vậy tốt nhất là nên tránh tình trạng mất cân bằng này.

Một chương trình tên balancer (chương trình này sẽ chạy như là một daemon trên NameNode) sẽ thực hiện việc cân bằng lại cluster. Việc khởi động hay mở chương trình này sẽ độc lập với HDFS (tức khi HDFS đang chạy, ta có thể tự do tắt hay mở chương trình này), tuy nhiên nó vẫn là một thành phần trên HDFS. Balancer sẽ định kỳ thực hiện phân tán lại các bản sao của block dữ liệu bằng các di chuyển nó từ các DataNode đã quá tải sang những DataNode còn trống mà vẫn đảm bảo các chiến lược sắp xếp bản sao của các block lên các DataNode như ở phần 2.2.3.2 .

#### 2.2.3.4 Thu nhặt rác (Gabage collettion)

Sau khi một file trên HDFS bị delete bởi người dùng hoặc ứng dụng, nó sẽ không lập tức bị xoá bỏ khỏi HDFS. Thay vào đó, đầu tiên HDFS sẽ đổi tên (rename) nó lại thành một file trong thư mục rác có tên /trash. Các tập tin sẽ được phục hồi nhanh 34 chóng nếu như nó vẫn còn ở trong thư mục rác. Sau một thời hạn 6 giờ (chúng ta có thể cấu hình thời hạn này lại), NameNode sẽ thực sự xoá file trong thư mục rác này đi. Việc xoá file kèm theo việc các bản sao của các block thuộc file đó sẽ thực sự bị xoá đi trên các DataNode.

Một người dùng có thể lấy lại tập tin bị xoá bằng cách vào thư mục /trash và di chuyển nó ra, miễn là nó vẫn chưa thực sự bị xoá khỏi /trash.

### 2.2.4 Khả năng chịu lỗi và chẩn đoán lỗi của HDFS

#### 2.2.4.1 Khả năng phục hồi nhanh chóng

Các NameNode và DataNode đều được thiết kế để có thể phục hồi nhanh chóng. Trong trường hợp NameNode ngừng hoạt động, ta chỉ cần phục hồi lại NameNode mà không cần phải restart tất cả các DataNode. Sau khi NameNode phục hồi nó sẽ tự động liên lạc lại với các DataNode và hệ thống lại phục hồi (thực chất là NameNode chỉ đứng yên và lắng nghe các HeartBeat từ các DataNode). Nếu một DataNode bất kỳ bị ngừng hoạt động, ta chỉ cần khởi động lại DataNode này và nó sẽ tự động liên lạc với NameNode thông qua các HeartBeat để cập nhật lại tình trạng của mình trên NameNode.

#### 2.2.4.2 Nhân bản các block

Như đã trình bày ở các phần trên, mỗi block dữ liệu trên HDFS được lưu trữ trùng lắp trên các DataNode khác nhau thuộc các rack khác nhau. Người dùng (hoặc ứng dụng) có thể gán các chỉ số mức độ nhân bản (replication level) khác nhau cho các file khác nhau, tuỳ vào mức độ quan trọng của file đó, chỉ số mặc định là ba. Nhờ vậy, khi một hay một số DataNode bị ngừng hoạt động, ta vẫn còn ít nhất một bản sao của block.

#### 2.2.4.3 Nhân bản metadata trên NameNode với SecondaryNameNode

Từ kiến trúc trên, ta thấy được tầm quan trọng của NameNode, nó lưu giữ tất cả các metadata của hệ thống. Nếu Namenode gặp phải sự cố gì đó (cả phần cứng hay phần mềm) thì tất cả các file trên hệ thống HDFS đều sẽ bị mất, vì ta không có cách nào để tái cấu trúc lại các file từ các block được lưu trên các DataNode. Đó là lý do có sự tồn tại của SecondaryNamenode. SecondaryNamenode là một node duy nhất trên Hadoop cluster. SecondaryNamenode không đóng vai trò như một NameNode, nhiệm vụ của SecondaryNamenode là lưu trữ lại checkpoint (trạng thái thống nhất của metadata) mới nhất trên NameNode. Khi NameNode gặp sự cố, thì checkpoint mới nhất này sẽ được import vào NameNode và NameNode sẽ trở lại hoạt động như thời điểm SecondaryNamenode tạo checkpoint. SecondaryNamenode thực hiện nhiệm vụ của nó thông qua một daemon tên secondarynamenode.

SecondaryNamenode không cập nhật checkpoint bằng cách tải toàn bộ metadata trên NameNode về. Thực chất, SecondaryNamenode chỉ tải phần EditLog từ NameNode về và thực hiện việc “trộn” EditLog này vào trong phiên bản metadata trước đó. Cấu trúc của metadata trên SecondaryNamenode cũng giống như cấu trúc metadata trên NameNode.

#### 2.2.4.4 Toàn vẹn dữ liệu trên HDFS

HDSF đảm bảo tính toàn vẹn của dữ liệu bằng cách thực hiện tạo checksum5 tất cả dữ liệu ghi lên nó và sẽ kiểm tra lại checksum mỗi khi đọc dữ liệu. HDFS sẽ tạo một checksum cho mỗi 512 bytes dữ liệu được ghi (ta có thể cấu hình lại tham số này thông qua property io.bytes.per.checksum, xem phụ lục).

DataNode chịu trách nhiệm kiểm tra tính toàn vẹn dữ liệu bằng cách kiểm tra checksum trước khi lưu trữ dữ liệu và checksum của nó. Điều này được thực hiện khi DataNode nhận được dữ liệu từ client hay từ các DataNode khác trong quá trình nhân bản các block thông qua data pile (xem phần 2.2.2.3.2. ). Khi client đọc dữ liệu từ các DataNode, client cũng sẽ thực hiện kiểm tra checksum và so sánh chúng với checksum lưu trên DataNode.

### 2.2.5 Các giao diện tương tác

2.2.5.1 Giao diện command line.

Đây là giao diện đơn giản nhất để tương tác với HDFS. HDFS cung cấp các shell để thao tác trên folder, file như tạo, xoá, di chuyển, rename, copy…Các shell này đều thao tác trên HDFS thông qua các URI có dạng hdfs:///

2.2.5.2 Giao diện java.

Hadoop được viết bằng Java. Vì vậy, tất các thao tác tương tác với HDFS đều được thực hiện thông qua các Java API. Các shell hình thành nên giao diện command line của HDFS cũng được code từ các Java API này. Thông qua các Java API của Hadoop, ta có thể dễ dàng phát triển các ứng dụng tương tác với HDFS giống như với các hệ thống file truyền thống khác.

2.2.5.3 Giao diện web.

Đây là giao diện cho phép ta dễ dàng nắm bắt được tình trạng hoạt động của HDFS, biết được danh sách các node đang hoạt động, tình trạng đĩa cứng trên từng node… Giao diện này còn cho phép ta browse các file trên HDFS và download các file. Tuy nhiên ta không thể tạo các thay đổi lên hệ thống (tạo, xoá, cập nhật file/thư mục…) từ giao diện này. Địa chỉ tương tác với HDFS: **http://:50070/ 37**

2.2.6 Quản trị HDFS

2.2.6.1 Permission

HDFS có một mô hình phân quyền tập tin và thư mục giống với POSIX (Portable Operating System Interface [for Unix]). Có ba loại quyền truy cập: quyền được phép đọc (r), quyền ghi (w), và quyền thực thi (x).

Quyền được phép đọc cho phép người dùng (hoặc ứng dụng) đọc nội dung của tập tin hay danh sách nội dung của một thư mục.

Quyền ghi được đòi hỏi khi ghi một file, hoặc với một thư mục, để tạo hoặc xóa các file/thư mục trong nó. Quyền thực thi không đươc áp dụng cho một file vì chúng ta không thể thực thi một file trên HDFS (không giống như POSIX).

Quyền thực thi một thư mục được yêu cầu khi người dùng cố gắng truy cập vào các file hay thư mục con của thư mục đó.

Mỗi file và thư mục có chủ sở hữu (owner), một nhóm (group), và chế độ (mode). Mode mô tả cho quyền truy cập của owner vào tập tin/thư mục, quyền truy cập của các thành viên thuộc group vào tập tin/thư mục và quyền truy cập của những những người dùng không phải owner và cũng không thuộc group vào tập tin/thư mục.

Khi truy cập vào HDFS, client đươc nhận diện người dùng (user name) và nhóm (group) của tiến trình trên client. Các client truy cập vào hệ thống từ xa, điều này làm cho client có thể trở thành một người sử dụng tùy tiện, đơn giản bằng cách tạo một tài khoản trên hệ thống từ xa. Vì vậy, quyền truy cập chỉ được sử dụng trong một cộng đồng hợp tác của người dùng, như là một cơ chế cho việc chia sẻ hệ thống tập tin và tránh vô tình làm mất mát dữ liệu, chứ không phải dành cho việc bảo mật các tài nguyên trong một môi trường thù địch.

Tuy nhiên, bất chấp những nhược điểm trên, việc kích hoạt chế độ kiểm tra quyền truy cập sẽ có ý nghĩa trong việc tránh tình cờ sửa đổi hoặc xóa các bộ phận đáng kể của hệ thống tập tin, bởi người dùng hoặc bởi các công cụ tự động hay các chương trình. Lưu ý là trên HDFS ta có thể kích hoạt hay tắt chế độ kiểm tra quyền truy cập đi. Khi chế độ kiểm tra quyền truy cập được kích hoạt, mọi thao tác truy cập vào file/thư mục điều sẽ được kiểm tra quyền hạn. Trên HDFS còn có một người dùng đặc biệt, đó là super-user. Đây chính là user đại diện cho các tiến trình trên NameNode. User này có quyền hạn toàn cục và sẽ không bị kiểm tra quyền truy cập.

2.2.6.2 Quản lý hạn ngạch (quotas)

HDFS cho phép người quản trị có thể thiết lập hạn ngạch (quotas) cho số lượng tên (file/thư mục) sử dụng và dung lượng sử dụng cho các thư mục. Có hai loại hạn ngạch là hạn ngạch tên (name quotas) và hạn ngạch dung lượng (space quotas). Hai loại hạn ngạch này hoạt động độc lập, nhưng việc quản trị và thực thi của hai loại hạn ngạch này lại ảnh hưởng chặt chẽ tới nhau.

Hạn ngạch tên của một thư mục là một giới hạn cứng về số lượng file và thư mục trong cây thư mục bắt nguồn từ thư mục đó. Việc tạo mới tập tin và thư mục sẽ thất bại nếu hạn ngạch bị vượt qua. Các nỗ lực để thiết lập một hạn ngạch vẫn sẽ thành công ngay cả khi thư mục sẽ vi phạm hạn ngạch mới. Một thư mục mới được tạo ra sẽ không được thiết lập hạn ngạch.

Hạn ngạch dung lượng của một thư mục là một giới hạn cứng về số byte được sử dụng bởi các tập tin trong cây thư mục bắt nguồn thư mục đó. Việc cấp phát các block cho các file sẽ thất bại nếu hạn ngạch bị vượt qua. Mỗi bản sao một block trong file làm tăng dung lượng của thư mực và đưa nó tới gần hạn mức hơn. Một thư mục mới 39 được tạo ra không có liên quan đến hạn ngạch vì nó không làm tăng dung lượng của thư mục cha.

2.3 MapReduce

2.3.1 Giới thiệu mô hình tính toán MapReduce

2.3.1.1 Nguyên nhân ra đời và lịch sử

Trước thời điểm Google công bố mô hình MapReduce, với sự bùng nổ của dữ liệu (hàng petrabyte), cùng lúc đó nhu cầu thực hiện xử lý các nghiệp vụ trên lượng dữ liệu khổng lồ là thách thức lớn lúc bấy giờ. Cùng với nhu cầu ấy, các doanh nghiệp đang gặp vấn đề tương tự khi muốn tìm một giải pháp tốn ít chi phí và hiệu năng thể hiện cao. Trong khi nghiên cứu, một nhóm nhân viên của Google đã khám phá ra một ý tưởng để giải quyết nhu cầu xử lý lượng dữ liệu lớn là việc cần phải có hệ thống nhiều các máy tính và cần có các thao tác để xử lý đồng bộ trên hệ thống đó. Và họ đã xác định được 2 thao tác cơ bản là Map và Reduce, nó được lấy cảm hứng từ phong cách lập trình hàm (Functional Programming). Với ý tưởng trên, Google đã phát triển thành công mô hình MapReduce, là mô hình dùng cho xử lý tính toán song song và phân tán trên hệ thống phân tán. Nói một cách đơn giản hơn, mô hình này sẽ phân rã từ nghiệp vụ chính (do người dùng muốn thể hiện) thành các công việc con để chia từng công việc con này về các máy tính trong hệ thống thực hiện xử lý một cách song song, sau đó thu thập lại các kết quả. Với mô hình này, các doanh nghiệp đã cải thiện được đáng kể về hiệu suất xử lý tính toán trên dữ liệu lớn, chi phí đầu tư rẻ và độ an toàn cao.

2.3.1.2 Mô hình MapReduce (Theo công bố của Google)

Theo tài liệu “MapReduce: Simplified Data Processing on Large Clusters” của Google, Google định nghĩa rằng: “MapReduce là mô hình lập trình và thực thi song song các xử lý và phát sinh các tập dữ liệu lớn”. Tuy nhiên, với định nghĩa như vậy, chúng ta chưa thật sự hiểu rõ được mô hình MapReduce là như thế nào.

Hình bên dưới (Hình 2-7: Mô hình MapReduce của GoogleError) mô tả rõ hơn về mô hình MapReduce (Tham khảo từ bài lab MapReduce: Simplified Data Processing on Large Clusters <http://labs.google.com/papers/mapreduce.html> )

**A picture containing text, diagram, plan, circle

Description automatically generated**

Hình ‑ Mô hình Mapreduce của google

Để hiểu rõ hơn, MapReduce là một mô hình được áp dụng trên một hệ thống các máy tính được kết nối với nhau và cài đặt chương trình MapReduce, và thường kèm theo nó là một hệ thống chia sẻ file phân tán. Với mô hình MapReduce, từ một công việc thì nó sẽ chia nhỏ thành các công việc con giống nhau và dữ liệu đầu vào cũng được chia nhỏ thành các mảnh dữ liệu nhỏ hơn. Điều đặc biệt nhất, để thực hiện các thao tác xử lý một cách song song và đồng thời, MapReduce sử dụng hai thao tác 41 chính cho việc thực thi công việc ban đầu từ người dùng là hàm map và hàm reduce, có thể hiểu một cách đơn giản là hàm map tiếp nhận mảnh dữ liệu input và thực hiện xử lý nào đó (đơn giản như là lọc dữ liệu, hoặc trích dữ liệu) để chuẩn bị dữ liệu làm đầu vào cho hàm reduce, hàm reduce thực hiện xử lý riêng của nó và trả ra cho người dùng một phần nhỏ kết quả cuối cùng của công việc, sau khi tất cả hàm reduce thực hiện người dùng sẽ có được toàn bộ kết quả của công việc. Tiếp theo phần xử lý, với số lượng công việc con và số lượng mảnh dữ liệu trên, đầu tiên, hệ thống MapReduce sẽ gửi từng công việc và từng mảnh dữ liệu đến các máy tính trong hệ thống để thực hiện, bản chất là thực hiện hàm map một cách song song. Sau khi thực hiện xong hết các công việc con thông qua việc thực hiện hàm map thì hệ thống sẽ bắt đầu thực hiện các hàm reduce để trả ra các kết quả cuối cùng cho người dùng. MapReduce quản lý quá trình thực thi công việc bằng việc định nghĩa một máy trong hệ thống đóng vai trò là master và các máy còn lại đóng vai trò của một worker (dựa trên kiến trúc masterslave). Master chịu trách nhiệm quản lý toàn bộ quá trình thực thi công việc trên hệ thống như :tiếp nhận công việc, phân rã công việc thành công việc con, và phân công các công việc con cho các worker. Còn worker chỉ làm nhiệm vụ thực hiện công việc con được giao (thực hiện hàm map hoặc hàm reduce. Phần cơ chế hoạt động cũng phần nào tương tự như phần 2.3.2.2.2.

Để hiểu rõ được mô hình MapReduce, chúng ta cần phải hiểu rõ vai trò của hai hàm map và reduce, chúng cũng được xem là phần xử lý quan trọng nhất trong mô hình MapReduce. Hai hàm này đều được người dùng định nghĩa tùy theo nhu cầu sử dụng

#### 2.3.1.3 Hàm Map

A purple rectangular sign with black text

Description automatically generated with medium confidence

Hình ‑ Hàm map

Người dùng đưa một cặp dữ liệu (key,value) làm input cho hàm map, và tùy vào mục đích của người dùng mà hàm map sẽ trả ra danh sách các cặp dữ liệu (intermediate key,value).

#### 2.3.1.4 Hàm Reduce

A purple rectangular sign with black text

Description automatically generated with medium confidence

Hình ‑ Hàm reduce

Hệ thống sẽ gom nhóm tất cả value theo intermediate key từ các output của hàm map, để tạo thành tập các cặp dự liệu với cấu trúc là (key, tập các value cùng key). Dữ liệu input của hàm reduce là từng cặp dữ liệu được gom nhóm ở trên và sau khi thực hiện xử lý nó sẽ trả ra cặp dữ liệu (key, value) output cuối cùng cho người dùng.

2.3.2 Hadoop MapReduce Engine

Hadoop đã giữ nguyên cơ chế của MapReduce của Google để cài đặt thành bộ máy thực thi MapReduce. Đây là một framework cho phép dễ dàng phát triển và triển khai các ứng dụng MapReduce.

2.3.2.1 Một số khái niệm: Job, Task, JodTracker, TaskTracker

Trong mô hình MapReduce của Hadoop, Hadoop định nghĩa MapReduce Job (job) là một đơn vị nghiệp vụ mà người dùng muốn thực hiện, kèm theo đó là dữ liệu input. Ví dụ như nghiệp vụ :

* Tính số lần xuất hiện của từng từ trong một tài liệu.
* Tạo tập các inverted index của các từ từ các tập tài liệu.

Như đã biết, mô hình Hadoop MapReduce sử dụng lại hoàn toàn mô hình MapReduce của Google. Một MapReduce Job sẽ được phân rã thành các công việc con nhỏ hơn, được định nghĩa là task. Và task này được gồm có 2 loại: map task và reduce task. Hiểu một cách đơn giản, map task là công việc con được thực thi thông qua việc sử dụng hàm map ,và tương tự với reduce task sẽ thực hiện hàm reduce. Điều này sẽ được làm rõ hơn ở mục “Cơ chế hoạt động của Hadoop MapReduce”Như đã định nghĩa ở mục “Kiến trúc của một Hadoop cluster” , để quản lý và thực thi MapReduce Job, Hadoop đưa ra 2 khái niệm JobTracker và TaskTracker:

* JobTracker: là một máy vật lý cài đặt Hadoop MapReduce (như là master của hệ thống), với vai trò tiếp nhận các yêu cầu thực thi các MapReduce job, phân chia job này thành các task và phân công cho các TaskTracker thực hiện, quản lý tình trạng thực hiện các task của TaskTracker và phân công lại nếu cần. JobTracker cũng quản lý danh sách các node TaskTracker và tình trạng của từng node thông qua hearbeat. Điều đặc biệt, Hadoop chỉ định hệ thống chỉ có tối đa một JobTracker
* TaskTracker: là một máy vật lý cài đặt Hadoop MapReduce (là các worker của hệ thống), với vai trò tiếp nhận task được JobTracker phân công và thực hiện nó. Và hệ thống được phép có nhiều TaskTracker

#### 2.3.2.2 Kiến trúc MapReduce Engine

2.3.2.2.1. Kiến trúc các thành phần (JobTracker, TaskTracker)

A diagram of a job

Description automatically generated with medium confidence

Hình ‑ Kiến trúc thành phần

Xét một cách trừu tượng, Hadoop MapReduce gồm 4 thành phần chính riêng biệt (Hình 2-10: Kiến trúc các thành phần):

* Client Program: Chương trình HadoopMapReduce mà client đang sử dụng và tiến hành chạy một MapReduce Job.
* JobTracker: Tiếp nhận job và đảm nhận vai trò điều phối job này, nó có vai trò như bộ não của Hadoop MapReduce. Sau đó, nó chia nhỏ job thành các task, tiếp theo sẽ lên lịch phân công các task (map task, reduce task) này đến các tasktracker để thực hiện. Kèm theo vai trò của mình, JobTracker cũng có cấu trúc dữ liệu riêng của mình để sử dụng cho mục đích lưu trữ, ví dụ như nó sẽ lưu lại tiến độ tổng thể của từng job, lưu lại trang thái của các TaskTracker để thuận tiện cho thao tác lên lịch phân công task, lưu lại địa chỉ lưu trữ của các output của các TaskTracker thực hiện maptask trả về.
* TaskTracker: Đơn giản nó chỉ tiếp nhận maptask hay reducetask từ JobTracker để sau đó thực hiện. Và để giữ liên lạc với JobTracker, Hadoop Mapreduce cung cấp cơ chế gửi heartbeat từ TaskTracker đến JobTracker cho các nhu cầu như thông báo tiến độ của task do TaskTracker đó thực hiện, thông báo trạng thái hiện hành của nó (idle, in-progress, completed).
* HDFS: là hệ thống file phân tán được dùng cho việc chia sẻ các file dùng trong cả quá trình xử lý một job giữa các thành phần trên với nhau.

2.3.2.2.2. Cơ chế hoạt động

Hình bên dưới (Hình 2-11: Cơ chế hoạt động của Hadoop MapReduce) mô tả cơ chế hoạt động tổng quát của HadoopMapReduce, mô tả rõ cả quá trình từ lúc ClientProgram yêu cầu thực hiện job đến lúc các TaskTracker thực hiện reduce task trả về các kết quả output cuối cùng.

A diagram of a task tracker

Description automatically generated with low confidence

Hình ‑Cơ chế hoạt động của Hadoop Mapreduce

Đầu tiên chương trình client sẽ yêu cầu thực hiện job và kèm theo là dữ liệu input tới JobTracker. JobTracker sau khi tiếp nhận job này, nó sẽ thông báo ngược về chương trình client tình trạng tiếp nhận job. Khi chương trình client nhận được thông báo nếu tình trạng tiếp nhận hợp lệ thì nó sẽ tiến hành phân rã input này thành các split 6 (khi dùng HDFS thì kích thước một split thường bằng với kích thước của một đơn vị Block trên HDFS) và các split này sẽ được ghi xuống HDFS. Sau đó chương trình client sẽ gửi thông báo đã sẵn sàng để JobTracker biết rằng việc chuẩn bị dữ liệu đã thành công và hãy tiến hành thực hiện job.

Khi nhận được thông báo từ chương trình client, JobTracker sẽ đưa job này vào một stack mà ở đó lưu các job mà các chương trình client yêu cầu thực hiện. Tại một thời điểm JobTracker chỉ được thực hiện một job. Sau khi một job hoàn thành hay bị block, JobTracker sẽ lấy job khác trong stack này (FIFO) ra thực hiện. Trong cấu trúc dữ liệu của mình, JobTrack có một job scheduler với nhiệm vụ lấy vị trí các split (từ HDFS do chương trình client tạo), sau đó nó sẽ tạo một danh sách các task để thực thi. Với từng split thì nó sẽ tạo một maptask để thực thi, mặc nhiên số lượng maptask bằng với số lượng split. Còn đối với reduce task, số lượng reduce task được xác định bởi chương trình client. Bên cạnh đó, JobTracker còn lưu trữ thông tin trạng thái và tiến độ của tất cả các task.

A picture containing text

Description automatically generated

Hình ‑ Sự liên lạc đầu tiên giữa TaskTracker thực thi Maptask và JobTracker

Ngay khi JobTracker khởi tạo các thông tin cần thiết để chạy job, thì bên cạnh đó các TaskTracker trong hệ thống sẽ gửi các heartbeat đến JobTracker. Hadoop cung cấp cho các TaskTracker cơ chế gửi heartbeat đến JobTracker theo chu kỳ thời gian7 nào đó, thông tin bên trong heartbeat này cho phép JobTrack biết được TaskTracker này có thể thực thi task hay không (Hình 2-12: Sự liên lạc đầu tiên giữa TaskTracker thực thi Maptask và JobTracker) . Nếu TaskTracker còn thực thi được thì JobTracker sẽ cấp task và vị trí split tương ứng đến TaskTracker này để thực hiện. Tại sao ở đây ta lại nói TaskTracker còn có thể thực thi task hay không. Điều này được lý giải là do một Tasktracker có thể cùng một lúc chạy nhiều map task và reduce task một cách đồng bộ, số lượng các task này dựa trên số lượng core, số lượng bộ nhớ Ram và kích thước heap8 bên trong TaskTracker này.

Việc TaskTracker thực thi task được chia thành 2 loại: TaskTracker thực thi maptask, TaskTracker thực thi reduce task.

A diagram of a computer

Description automatically generated with low confidence

Hình ‑ Cơ chế hoạt động của MapTask

Khi một TaskTracker nhận thực thi maptask, kèm theo đó là vị trí của input split trên HDFS. Sau đó, nó sẽ nạp dữ liệu của split từ HDFS vào bộ nhớ, rồi dựa vào kiểu format của dữ liệu input do chương trình client chọn thì nó sẽ parse split này để phát sinh ra tập các record, và record này có 2 trường: key và value. Cho ví dụ, với kiểu input format là text, thì tasktracker sẽ cho phát sinh ra tập các record với key là offset đầu tiên của dòng (offset toàn cục), và value là các ký tự của một dòng. Với tập các record này, tasktracker sẽ chạy vòng lặp để lấy từng record làm input cho hàm map để trả ra out là dữ liệu gồm intermediate key9 và value. Dữ liệu output của hàm map sẽ ghi xuống bộ nhớ chính, và chúng sẽ được sắp xếp trước ngay bên trong bộ nhớ chính (Hình 2-13: Cơ chế hoạt động của Map task).

A diagram of a job tracker

Description automatically generated with low confidence

Hình ‑ Tasktracker hoàn thành Maptask

Trước khi ghi xuống local disk, các dữ liệu output này sẽ được phân chia vào các partition (region) dựa vào hàm partition10, từng partition này sẽ ứng với dữ liệu input của reduce task sau này. Và ngay bên trong từng partition, dữ liệu sẽ được sắp xếp (sort) tăng dần theo intermediate key, và nếu chương trình client có sử dụng hàm combine11 thì hàm này sẽ xử lý dữ liệu trên từng partition đã sắp xếp rồi. Sau khi thực hiện thành công maptask thì dữ liệu output sẽ là các partition được ghi trên local, ngay lúc đó TaskTracker sẽ gửi trạng thái completed của maptask và danh sách các vị trí của các partition output trên localdisk của nó đến JobTracker(Hình 2-14: TaskTracker hoàn thành Map task). Đó là toàn bộ quá trình TaskTracker thực hiện một maptask

A diagram of a task tracker

Description automatically generated with low confidence

Hình ‑ Cơ chế hoạt động của ReduceTask

Khác với TaskTracker thực hiện maptask, TaskTracker thực hiện reduce task theo một cách khác. TaskTracker thực hiện reduce task với dữ liệu input là danh sách các vị trí của một region cụ thể trên các output được ghi trên localdisk của các maptask. Điều này có nghĩa là với region cụ thể, JobTracker sẽ thu thập các region này trên các output của các maptask thành một danh sách các vị trí của các region này (Hình 2-15: Cơ chế hoạt động của Reduce task).

Do biết được số lượng map task và reduce task, nên TaskTracker một cách định kỳ sẽ hỏi JobTracker về các vị trí của region mà sẽ phân bổ cho nó cho tới khi nó nhận 52 được đầy đủ các vị trí region của output của tất cả các map task trong hệ thống.Với danh sách vị trí này, TaskTracker sẽ nạp12 (copy) dữ liệu trong từng region ngay khi map task mà output chứa region này hoàn thành vào trong bộ nhớ. Và TaskTracker này cũng cung cấp nhiều tiểu trình thực hiện nạp dữ liệu đồng thời để gia tăng hiệu suất xử lý song song.

Sau khi nạp thành công tất cả các region thì TaskTracker sẽ tiến hành merge13 dữ liệu của các region theo nhiều đợt mà các đợt này được thực hiện một cách đồng thời để làm gia tăng hiệu suất của thao tác merge. Sau khi các đợt merge hoàn thành sẽ tạo ra các file dữ liệu trung gian được sắp xếp. Cuối cùng các file dữ liệu trung gian này sẽ được merge lần nữa để tạo thành một file cuối cùng. TaskTracker sẽ chạy vòng lặp để lấy từng record ra làm input cho hàm reduce, hàm reduce sẽ dựa vào kiểu format của output để thực hiện và trả ra kết quả output thích hợp. Tất cả các dữ liệu output này sẽ được lưu vào một file và file này sau đó sẽ được ghi xuống HDFS.

A diagram of a job tracker

Description automatically generated with low confidence

Hình ‑ Task tracker hoàn thành Reduce Task

Khi TaskTracker thực hiện thành công reduce task, thì nó sẽ gửi thông báo trạng thái “completed” của reduce task được phân công đến JobTracker. Nếu reduce task này là task cuối cùng của job thì JobTracker sẽ trả về cho chương trình người dùng biết job này đã hoàn thành (Hình 2-16: TaskTracker hoàn thành Reduce task). Ngay lúc đó JobTracker sẽ làm sạch cấu trúc dữ liệu của mình mà dùng cho job này, và thông báo cho các TaskTracker xóa tất cả các dữ liệu output của các map task (Do dữ liệu maptask chỉ là dữ liệu trung gian làm input cho reduce task, nên không cần thiết để lưu lại trong hệ thống).

2.3.2.2.3. MapReduce và HDFS (Các đặc điểm tối ưu của MapReduce khi kết hợp với HDFS)

Như đã biết HDFS chỉ là hệ thống file phân tán với các cơ chế quản lý bên trong nó. Vậy tại sao người ta lại đưa ra việc kết hợp giữa MapReduce và HDFS?

Thứ nhất, MapReduce đơn thuần làm nhiệm vụ xử lý tính toán song song, vậy trong một hệ thống phân tán thì dữ liệu sẽ được kiểm soát như thế nào để người dùng có thể dễ dàng truy xuất, do đó việc sử dụng HDFS cho việc bổ các input split của MapReduce xuống và có kích thước gần bằng với kích thước block, đều này làm tăng hiệu suất cho việc xử lý song song và đồng bộ của các TaskTracker với từng split mà có thể xử lý riêng biệt này. Thêm vào đó, các dữ liệu output cuối cùng của một MapReduce Job cũng được lưu trữ xuống HDFS, đều này giúp cho người dùng tại một máy tính nào đó trong hệ thống đều có thể lấy được toàn bộ kết quả output này thông qua các phương thức thuộc cơ chế quản lý của HDFS (Tính trong suốt). Bên cạnh đó, khi các block không đặt tình trạng cân bằng (load-balancer) thì HDFS có cơ chế thực hiện việc cân bằng các block trở lại một cách hiệu quả, điều này sẽ làm gia tăng hiệu suất của data locality (được nói ở ngay bên dưới)

A picture containing text, screenshot, diagram, parallel

Description automatically generated

Hình ‑ Data Locality

Thứ hai, với việc các input split được bổ phân tán trên toàn hệ thống, thì HDFS cho phép các JobTracker biết được liệu một input split và các replica (bản sao được tạo bởi HDFS) đang được lưu trữ trên một máy vật lý nào. Điều này thật sự quan trọng vì nếu biết được thông tin này, thì JobTracker sẽ phân bổ cho các TaskTracker thực hiện maptask với replica mà định vị ngay bên trong máy tính đang làm nhiệm vụ TaskTracker. Việc này sẽ làm TaskTracker không phải tốn chi phi về thời gian để nạp dữ liệu từ các máy tính khác, do không phải sử dụng tới băng thông mạng của hệ thống. Với cơ chế trên, thì MapReduce sẽ gia tăng được hiệu suất thể hiện về mặt thời gian, đây là một cải tiến rất cần trong hệ thống phân tán. Cơ chế này được Google và 56 Hadoop định nghĩa là data locality (Hình 2-17: Data locality). Cơ chế data locality sẽ đem về hiệu suất khác biệt với hệ thống lớn vì không phải tiêu thụ băng thông mạng cho việc vận chuyển qua lại dữ liệu giữa các máy tính vật lý. Bên cạnh đó, nếu replica mà không nằm trong một máy tính TaskTracker, thì JobTracker sẽ phân bố một replica mà nằm trong một máy mà thuộc cùng một switch mạng (Trong các hệ thống lớn, người ta có thể gom nhóm các TaskTracker thành một rack, và rack này được kết nối với nhau thông qua switch, và switch này cũng được kết nối với các switch tương tự), điều này cũng giảm đáng kể chi phí đọc dữ liệu từ xa và tiêu thụ băng thông.

2.3.2.3 Phát triển ứng dụng theo mô hình MapReduce với Hadoop MapReduce

Sau đây là toàn bộ quá trình phát triện một ứng dụng theo mô hình MapReduce với HadoopMapReduce (Hình 2-18: Phát triển ứng dụng MapReduce trên Hadoop).

A picture containing text, diagram, parallel, line

Description automatically generated

Hình ‑ Phát triển ứng dụng Mapreduce trên Hadoop

Quá trình phát triển được phân rõ ra theo công việc nào do người dùng thực hiện can thiệp và công việc nào bên trong framework tự làm.

Đối với người dùng, họ chỉ can thiệp vào việc phát triển ứng dụng qua các giai đoạn sau:

* Thiết lập thông số cấu hình của hệ thống cũng như của MapReduce Job.
* Tiếp theo, truyền vào kiểu format cho cách thức đọc file (như file text, file kết hợp, hoặc file Database), tiếp theo là kiểu format của dữ liệu input, điều này thật sự có ý nghĩa với việc sử dụng hàm map, vì với từng kiểu format này mà từ đó với các split sẽ cho ra tập các record với giá trị key và value khác nhau.
* Sau đó người dùng phải truyền vào đường dẫn của dữ liệu input.
* Kế tiếp, một trong 2 phần việc quan trọng nhất là định nghĩa hàm map để từ đó cho ra được kết quả output trung gian như ý muốn. Và để dữ liệu output của maptask đúng format để reduce task thực hiện thì người dùng phải chọn kiểu format nào cho key và kiểu format nào cho value của từng record output của hàm map.
* Rồi kế đến là công việc không kém phần quan trọng và có ý nghĩa lớn với dữ liệu output của maptask là công việc thiết lập thông tin về số lượng reduce task. Từ thông tin này mà hàm partition mới có cơ sở để thực hiện.
* Kế đến, công việc quan trọng còn lại trong 2 công việc quan trọng được nhắc trước đó (hàm map) là hàm reduce. Thêm vào đó người dùng chọn kiểu format cho từng record (key, value) của dữ liệu output của hàm reduce.
* Cuối cùng là công việc người dùng chọn kiểu format cho dữ liệu output cuối cùng (ví dụ như: output thành nhiều file) và vị trí mà file output sẽ lưu.

Sau đây là thứ tự công việc mà hệ thống (do framework làm) thực hiện quá trình phát triển ứng dụng:

* Với thông tin cấu hình, hệ thống sẽ kiểm tra các thông tin này liệu có hợp lệ hay không, để sau đó mới thông báo người dùng là ứng dụng có thể bắt đầu
* Sau đó, hệ thống sẽ dựa vào 2 thông tin về kiểu format (format đọc dữ liệu và format từng record input) và đường dẫn dữ liệu input trên để tiến hành tính toán và thực hiện việc chia nhỏ dữ liệu input này thành các input split.
* Sau khi có được tập dữ liệu input split, thì hệ thống phân tán map task trên các TaskTracker thực hiện.
* Hàm map sẽ trả về từng record với format như người dùng định nghĩa, và với tập record này hệ thống sẽ thực hiện thao tác phân chia chúng vào từng vào từng partition (số lượng partition đúng bằng số lượng reduce task), kế đến thao tác sắp xếp theo khóa sẽ được thực hiện trong từng partition.
* Sau khi có được tập các dữ liệu output của các maptask, hệ thống thực hiện một thao tác để lấy dữ liệu của một partition trên các ouput của maptask, để sau đó hệ thống sẽ thực hiện thao tác trộn các dữ liệu này lại, rồi tiến hành thao tác sắp xếp để cho ra tập các record (key, danh sách value) để rồi từ đó chạy hàm reduce task.
* Reduce task sẽ trả ra từng record với kiểu format đã được người dùng định nghĩa trước. Với kiểu format của dữ liệu output cuối cùng và đường dẫn của mà file được lưu thì reduce task sẽ lưu dữ liệu output theo đúng 2 thông tin trên.

2.3.2.4 Ứng dụng của MapReduce

MapReduce không phải là mô hình áp dụng được cho tất cả mọi vấn đề. Thực tế, mô hình MapReduce áp dụng tốt được cho các trường hợp cần xử lý một khối dữ liệu lớn bằng cách chia nó ra thành các mảnh nhỏ hơn và xử lý song song. Một số trường hợp sau sẽ thích hợp với MapReduce:

* Dữ liệu cần xử lý lớn, kích thước tập tin lớn.
* Các ứng dụng thực hiện xử lý, phân tích dữ liệu, thời gian xử lý đáng kể, có thể tính bằng phút, giờ, ngày, tháng.
* Cần tối ưu hoá về băng thông trên cluster.

Một số trường hợp có thể không phù hợp:

* Dữ liệu cần xử lý là tập hợp nhiều tập tin nhỏ.
* Cần tìm kiếm nhanh (tốc độ có ý nghĩa đến từng giây) trên tập dữ liệu lớn. Do độ trễ khi xử lý các MapReduce Job và khởi tạo các task trên DataNode.

# Thuật toán phân cụm Kmean kết hợp mô hình Mapreduce

## 3.1. Kmean là gì ?

### 3.1.1 Học không giám sát

Học không giám sát (Unsupervised Learning) là một phương pháp trong Machine Learning, trong đó mô hình được huấn luyện từ dữ liệu không có nhãn hoặc thông tin giám sát nào. Nói cách khác, mô hình không được chỉ dẫn hoặc hướng dẫn bằng cách sử dụng các ví dụ được gắn nhãn trước đó. Thay vào đó, mô hình phải tự tìm ra các mẫu và cấu trúc trong dữ liệu để phân tích và tạo ra các kết quả.

Các thuật toán học không giám sát thường được sử dụng để phân cụm (clustering) dữ liệu, giúp tìm ra các nhóm dữ liệu có tính chất tương tự nhau. Nó cũng được sử dụng để giảm chiều dữ liệu (dimensionality reduction), giúp giảm số chiều của dữ liệu mà vẫn giữ lại các thông tin quan trọng. Học không giám sát còn được sử dụng trong việc tìm kiếm các quy tắc ẩn (association rules) trong dữ liệu, giúp phát hiện các mối liên hệ giữa các đặc trưng.

Tuy nhiên, học không giám sát cũng có những hạn chế, như là không thể đưa ra các dự đoán chính xác về các nhãn của dữ liệu, không thể giải thích được các kết quả một cách rõ ràng và khó khăn trong việc đánh giá chất lượng của mô hình.

Trong thuật toán K-means clustering, chúng ta không biết nhãn (label) của từng điểm dữ liệu. Mục đích là làm thể nào để phân dữ liệu thành các cụm (cluster) khác nhau sao cho dữ liệu trong cùng một cụm có tính chất giống nhau.

### 3.1.2 Giới thiệu thuật toán Kmean

Ví dụ: Một công ty muốn tạo ra những chính sách ưu đãi cho những nhóm khách hàng khác nhau dựa trên sự tương tác giữa mỗi khách hàng với công ty đó (số năm là khách hàng; số tiền khách hàng đã chi trả cho công ty; độ tuổi; giới tính; thành phố; nghề nghiệp; …). Giả sử công ty đó có rất nhiều dữ liệu của rất nhiều khách hàng nhưng chưa có cách nào chia toàn bộ khách hàng đó thành một số nhóm/cụm khác nhau. Nếu một người biết Machine Learning được đặt câu hỏi này, phương pháp đầu tiên anh (chị) ta nghĩ đến sẽ là K-means Clustering. Vì nó là một trong những thuật toán đầu tiên mà anh ấy tìm được trong các cuốn sách, khóa học về Machine Learning. Và tôi cũng chắc rằng anh ấy đã đọc blog Machine Learning cơ bản. Sau khi đã phân ra được từng nhóm, nhân viên công ty đó có thể lựa chọn ra một vài khách hàng trong mỗi nhóm để quyết định xem mỗi nhóm tương ứng với nhóm khách hàng nào. Phần việc cuối cùng này cần sự can thiệp của con người, nhưng lượng công việc đã được rút gọn đi rất nhiều.

Ý tưởng đơn giản nhất về cluster (cụm) là tập hợp các điểm ở gần nhau trong một không gian nào đó (không gian này có thể có rất nhiều chiều trong trường hợp thông tin về một điểm dữ liệu là rất lớn). Hình bên dưới là một ví dụ về 3 cụm dữ liệu (từ giờ tôi sẽ viết gọn là cluster).

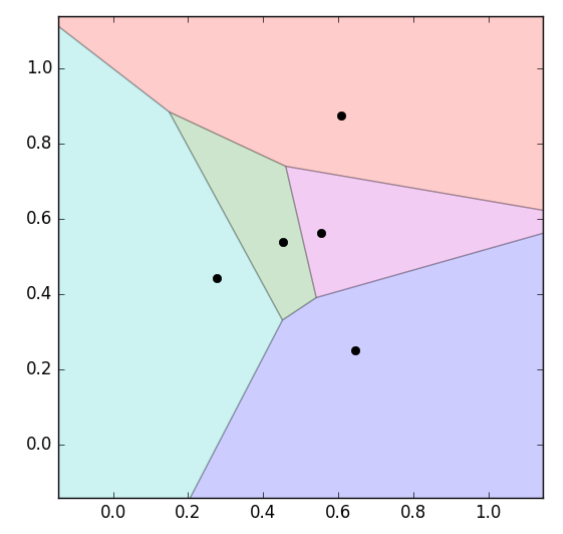
A picture containing colorfulness, screenshot

Description automatically generated

Hình ‑ Mô tả bài toán phân cụm với 3 cluster

Giả sử mỗi cluster có một điểm đại diện (center) màu vàng. Và những điểm xung quanh mỗi center thuộc vào cùng nhóm với center đó. Một cách đơn giản nhất, xét một điểm bất kỳ, ta xét xem điểm đó gần với center nào nhất thì nó thuộc về cùng nhóm với center đó. Tới đây, chúng ta có một bài toán thú vị: Trên một vùng biển hình vuông lớn có ba đảo hình vuông, tam giác, và tròn màu vàng như hình trên. Một điểm trên biển được gọi là thuộc lãnh hải của một đảo nếu nó nằm gần đảo này hơn so với hai đảo kia . Hãy xác định ranh giới lãnh hải của các đảo.

Hình dưới đây là một hình minh họa cho việc phân chia lãnh hải nếu có 5 đảo khác nhau được biểu diễn bằng các hình tròn màu đen:



Hình ‑ Phân vùng lãnh hải của mỗi đảo, các vùng khác nhau có màu sắc khác nhau

Chúng ta thấy rằng đường phân định giữa các lãnh hải là các đường thẳng (chính xác hơn thì chúng là các đường trung trực của các cặp điểm gần nhau). Vì vậy, lãnh hải của một đảo sẽ là một hình đa giác.

Cách phân chia này trong toán học được gọi là Voronoi Diagram.

Trong không gian ba chiều, lấy ví dụ là các hành tinh, thì (tạm gọi là) lãnh không của mỗi hành tinh sẽ là một đa diện. Trong không gian nhiều chiều hơn, chúng ta sẽ có những thứ (mà tôi gọi là) siêu đa diện (hyperpolygon).

Quay lại với bài toán phân nhóm và cụ thể là thuật toán K-means clustering, chúng ta cần một chút phân tích toán học trước khi đi tới phần tóm tắt thuật toán ở phần dưới. Nếu bạn không muốn đọc quá nhiều về toán, bạn có thể bỏ qua phần này. (Tốt nhất là đừng bỏ qua, bạn sẽ tiếc đấy).

### 3.1.3 Phân tích toán học

Mục đích cuối cùng của thuật toán phân nhóm này là: từ dữ liệu đầu vào và số lượng nhóm chúng ta muốn tìm, hãy chỉ ra center của mỗi nhóm và phân các điểm dữ liệu vào các nhóm tương ứng. Giả sử thêm rằng mỗi điểm dữ liệu chỉ thuộc vào đúng một nhóm.

Giả sử có N điểm dữ liệu là và K < N là số cluster chúng ta muốn phân chia. Chúng ta cần tìm các center và label của mỗi điểm dữ liệu.

Lưu ý về ký hiệu toán học: trong các bài viết của tôi, các số vô hướng được biểu diễn bởi các chữ cái viết ở dạng không in đậm, có thể viết hoa, ví dụ

Các ma trận được biểu diễn bởi các chữ viết hoa in đậm, ví dụ **X,M,Y.**

Với mỗi điểm dữ liệu là label vector của nó, trong đó nếu xi  được phân vào cluster k thì Điều này có nghĩa là có đúng một phần tử của vector **y­**i là bằng 1 (tương ứng với cluster của **x**i ) , các phần tử còn lại bằng 0. Ví dụ: nếu một điểm dữ liệu có label vector là [1,0,0,0,0..0] thì nó thuộc vào cluster 1, là [0,1,0,0,0,0] thì nó thuộc vào cluster 2, …Cách mã hóa label của dữ liệu như thế này được gọi là biểu diễn **one-hot**

Ràng buộc của yi, có thể viết dưới dạng toán học như sau:

Hàm mất mát và bài toán tối ưu:

Nếu ta coi center **mk** là center (hoặc representative) của mỗi cluster và ước lượng tất cả các điểm được phân vào cluster này bởi **mk**  thì một điểm dữ liệu xi được phân vào cluster k sẽ bị sai số là

Chúng ta mong muốn sai số này có trị tuyệt đối nhỏ nhất nên ta sẽ tìm cách để đại lượng sau đây đạt giá trị nhỏ nhất:

Hơn nữa, vì xi được phân vào cluster k nên . Khi đó biểu thức sẽ được viết lại là:

Sai số cho toàn bộ dữ liệu sẽ là:

Trong đó lần lượt là các ma trận được tạo bởi label vector của mỗi điểm dữ liệu và center của mỗi cluster. Hàm số mất mát trong bài toán K-means clustering của chúng ta là hàm L(Y,M) với ràng buộc như được nêu trong phương trình (1)

Tóm lại, chúng ta cần tối ưu bài toán sau:

Thỏa mãn điều kiện:

Tóm tắt lại thuật toán:

A picture containing text, diagram, font, line

Description automatically generated

Hình ‑ Lưu đồ thuật toán Kmean

* Đầu vào: Dữ liệu **X** là số lượng cluster **K**
* Đầu ra: Các center **M** và label vector cho từng điểm dữ liệu **Y**

1. Chọn K điểm bất kỳ làm các center ban đầu
2. Phân mỗi điểm dữ liệu vào cluster có center gần nó nhất.
3. Nếu việc gán dữ liệu vào từng cluster ở bước 2 không thay đổi so với vòng lặp trước nó thì ta dừng thuật toán.
4. Cập nhật center cho từng cluster bằng cách lấy trung bình cộng của tất các các điểm dữ liệu đã được gán vào cluster đó sau bước 2.
5. Quay lại bước 2.

A picture containing diagram, line, text, screenshot

Description automatically generated

Hình ‑ Minh họa quá trình phân cụm

Chúng ta có thể đảm bảo rằng thuật toán sẽ dừng lại sau một số hữu hạn vòng lặp. Thật vậy, vì hàm mất mát là một số dương và sau mỗi bước 2 hoặc 3, giá trị của hàm mất mát bị giảm đi. Theo kiến thức về dãy số trong chương trình cấp 3: nếu một dãy số giảm và bị chặn dưới thì nó hội tụ! Hơn nữa, số lượng cách phân nhóm cho toàn bộ dữ liệu là hữu hạn nên đến một lúc nào đó, hàm mất mát sẽ không thể thay đổi, và chúng ta có thể dừng thuật toán tại đây.

### 3.1.4 Điều kiện dừng thuật toán

Có hai điều kiện dừng chính của thuật toán K-means:

1. Số lần lặp tối đa: Thuật toán sẽ dừng lại sau khi thực hiện một số lần lặp được chỉ định trước đó. Điều này giúp tránh việc thuật toán chạy vô hạn hoặc quá lâu.
2. Sự hội tụ: Nếu các centroid không di chuyển nữa, có nghĩa là việc gán nhãn cho các điểm dữ liệu đã ổn định và thuật toán đã hội tụ. Việc này được xác định bằng cách so sánh tọa độ của centroid hiện tại và trước đó để xem chúng có khác biệt đáng kể hay không.

Nếu một trong hai điều kiện này được đáp ứng, thuật toán K-means sẽ dừng lại.

### 3.15. Chỉ số đánh giá Davies-Bouldin

[6]Từ thống kê cho đến Data mining, khai phá dữ liệu, từ các kết luận về tổng thể nghiên cứu, cho đến việc xây dựng các mô hình phân tích, tất cả đều phải được đánh giá một cách cụ thể, đây là bước quan trọng và yếu tố cốt lõi đóng góp vào khả năng thành công trong việc ứng dụng phân tích dữ liệu vào thực tế. Do đó, Clustering cũng không phải ngoại lệ.

Nói về các phương pháp đánh giá các phân cụm tìm được khi thực hiện những thuật toán clustering thì khó có thể kể hết, và có nhiều hướng tiếp cận khác nhau, phụ thuộc vào loại dữ liệu (định tính, định lượng), phụ thuộc vào loại clustering (ví dụ hierarchical hay k-means clustering), phụ thuộc vào thông tin có được (chính tập dữ liệu, dữ liệu bên ngoài),…

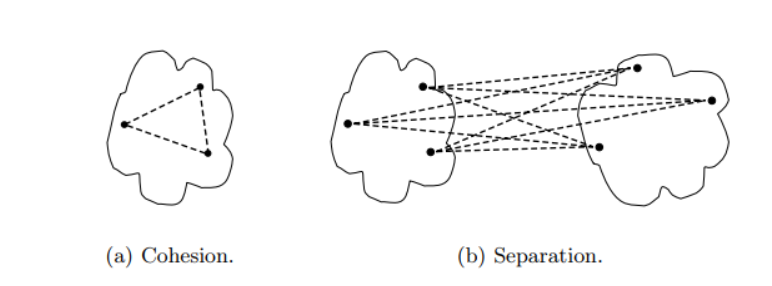
Đã có hàng loạt các chuyên gia nghiên cứu để tìm ra các phương pháp đánh giá tiêu chuẩn, đúng nhất, phù hợp nhất có thể áp dụng được cho mọi trường hợp triển khai clustering. Tuy nhiên hiện tại vẫn chưa khẳng định liệu có một cách thức chung, nhất quán, tốt nhất hay không.

Để có cái nhìn tổng quan về phương pháp đánh giá clustering chúng ta phải tìm hiểu chi tiết mục đích tại sao phải kiểm tra các clusters có được từ mô hình thuật toán phân cụm, ví dụ thông qua các câu hỏi sau:

* Các cluster tìm được có tương ứng hay đúng với thực tế hay đơn giản chỉ là giả thuyết, suy luận ở góc độ toán học?
* Chính xác thì sẽ có bao nhiêu cụm hay số cụm được phân tối ưu là bao nhiêu? Cái này thì chúng ta đã nói ở phần trên, elbow method.
* Giữa 2 kết quả phân cụm A và B, mỗi kết quả sẽ có cách phân cụm và số cụm khác nhau, thì kết quả nào sẽ phù hợp hơn để ứng dụng vào thực tế.
* Xác định khuynh hướng phân cụm (clustering tendency) tức là tìm hiểu xem cấu trúc của mỗi cụm tìm được có phi ngẫu nhiêu hay theo một quy tắc nào đó không?
* Đo lường chất lượng của mỗi phương pháp phân cụm áp dụng, thông qua kết quả phân tích có được, xác định: các clusters có “fit” với dữ liệu hay không (mô hình hoạt động tốt trên training data thì khi sử dụng testing data có vấn đề gì không), các cluster tìm được và trong thực tế có giống nhau không, các đối tượng thuộc các cluster tìm được trong thực tế có được phân loại đúng như vậy hay không (sử dụng dữ liệu classified trong thực tế nếu có để so sánh)

Cũng từ các câu hỏi trên mà những phương pháp đánh giá clusters được chia làm 2 loại chính Unspervised và Supervised nhưng trong bài này chúng ta chỉ tìm hiểu về Unsupervised. các phương pháp đánh giá độ hiệu quả, sự phù hợp của các clusters mà không sử dụng các thông tin bên ngoài tập dữ liệu, chỉ sử dụng những thông tin có sẵn trong tập dữ liệu. Thường được chia làm 2 loại:

* các phương pháp đo lường tính liên kết trong cụm (cluster cohesion – độ gắn kết, chặt chẽ), xác định mức độ liên quan chặt chẽ của các đối tượng trong một cụm
* và các phương pháp phân đo lường tính tách biệt giữa các cụm cụm (cluster seperation – cô lập), xác định mức độ khác biệt hoặc tách biệt của một cụm so với các cụm khác.



Hình ‑ Minh họa cách đánh giá

Chỉ số DBI là một công thức được sử dụng để đánh giá xem các nhóm (hay cụm) của các số liệu có được gom lại tốt hay không.Tưởng tượng bạn có một bộ số liệu và bạn muốn chia thành nhiều nhóm khác nhau, sao cho các số liệu trong cùng một nhóm rất giống nhau, trong khi các nhóm khác nhau khác biệt lớn. Điều này giúp cho việc phân tích và hiểu được bộ số liệu trở nên dễ dàng hơn.

Chỉ số DBI sẽ giúp bạn đánh giá xem những nhóm này đã được gom tốt hay chưa, bằng cách tính khoảng cách giữa các điểm trong cùng một nhóm và khoảng cách giữa các nhóm khác nhau. Nếu khoảng cách giữa các nhóm khác nhau lớn và khoảng cách giữa các điểm trong cùng một nhóm nhỏ thì chỉ số DBI sẽ nhỏ và coi như việc gom nhóm là tốt. Ngược lại, nếu khoảng cách giữa các nhóm khác nhau nhỏ hoặc khoảng cách giữa các điểm trong cùng một nhóm lớn, thì chỉ số DBI cao, cho thấy việc gom nhóm chưa tốt.

Đầu tiên, tính toán trung bình cục bộ (centroid) của mỗi cụm:

Trong đó:

* Ci là trung bình cục bộ của cụm thứ i
* Si  là tập hợp các điểm trong cụm thứ i
* xj là điểm thứ j

Tiếp theo, tính toán khoảng cách giữa hai trung bình cục bộ khác nhau:

2

Sau đó, tính toán một giá trị gọi là "scatter" (Si) cho từng cụm:

Cuối cùng, tính toán chỉ số DBI cho toàn bộ các cụm:

Trong đó:

* k là số lượng cụm
* i và j là chỉ số của các cụm trong tập hợp các cụm
* d(Ci , Cj) là khoảng cách giữa hai trung bình cục bộ khác nhau
* Si  và Sj là scatter của cụm thứ i và thứ j tương ứng

### 3.2. Kmean sử dụng Mapreduce

### 3.2.1 Nhận xét thuật toán Kmean

* Phần lớn khối lượng tính toán tập trung ở bước 2: tính khoảng cách từ mỗi điểm ( đối tượng) tới các tâm cụm
* Số lượng đối tượng trong tập dữ liệu càng lớn, thời gian cần cho bước này càng nhiều
* Việc tính toán khoảng cách từ một điểm tới tâm cụm là độc lập và không phụ thuộc vào điểm khác

🡺Từ đó ta có thể tính khoảng cách từ các điểm đến các tâm cụm có thể thực hiện song song, đồng thời , từ đó giúp tăng hiệu suất thực thi chương trình

### 3.2.2 Ý tưởng kết hợp mapreduce

* Tách dữ liệu thành các nhóm nhỏ
* Với mỗi vòng lặp
  + - Map: Phân cụm trên từng nhóm nhỏ dữ liệu, với mỗi điểm dữ liệu sẽ tìm trọng tâm gần nhất
    - Tất cả dữ liệu được gom theo từng tâm cụm
    - Reduce: Tính tâm mới của các dữ liệu được gom (theo từng tâm cụm)

### 3.2.3 Giải pháp Mapreduce hóa Kmean

A picture containing diagram, text, plan, technical drawing

Description automatically generated

Hình ‑ Lưu đồ Mapreduce thuật toán Kmean

1. Biểu diễn dữ liệu:

* Dữ liệu lưu trữ dưới dạng list các hàng
* Mỗi hàng là list giá trị các thành phần của vector biểu diễn cho một điểm dữ liệu

1. Lưu trữ phân tán dữ liệu: do các điểm dữ liệu được tính toán độc lập với nhau => có thể lưu trữ các thành phần của dữ liệu trên nhiều máy khác nhau để có thể xử lý song song và tăng tốc tính toán
2. Trong mỗi vòng lặp:

* B1:Tính khoảng cách của mỗi điểm dữ liệu trong phần dữ liệu của nó tới các trọng tâm
* B2: Kiểm tra xem điểm đó gần trọng tâm nào nhất
* B3:Gom các điểm thuộc cùng một cụm lại để tính lại trọng tâm sau mỗi vòng lặp

Dữ liệu cần phân cụm là danh sách các hàng(có thể lưu trên file txt) được chuyển sang kiểu key- value làm đầu vào cho thuật toán

Mô hình cơ bản của Mapreduce:

Map(keyIn, ValIn) -> List(keyIn,ValIn)

Reduce(keyIn, List(valIn)) -> List(KeyOut,ValOut)

### 3.2.4 Hàm map\_Kmean

* Đầu vào: Cặp key-value biểu diễn tọa độ của một điểm
  + KeyIn: là giá trị byte offset của dòng
  + ValIn: là vector biểu diễn tọa độ của một điểm dữ liệu
* Xử lý:
  + Tính khoảng cách của điểm tới các trọng tâm ( trọng tâm hiện tại chưa phải là trọng tâm cuối cùng cần tìm)
  + Chuyển về cụm có tâm gần nhất
* Đầu ra: Cặp key-value trung gian
  + KeyIn là tâm gần nhất( Trọng tâm hoặc chỉ số tâm)
  + ValIn: là tọa độ điểm thuộc cụm có trọng tâm là KeyIn

### 3.2.5 Hàm Reduce\_Kmean

* Trước khi thực hiện hàm reduce:
  + Kết quả của hàm map được trộn lại
  + Các cặp cùng KeyIn được gom thành một nhóm
* Đầu vào:
  + KeyIn được chuyển từ hàm map
  + List(valIn) là list các điểm valInt thuộc về cụm KeyIn
* Xử lý:
  + Tính trung bình cộng từng thành phần của các điểm cùng cụm
  + Cập nhật lại trọng tâm cụm đó
* Đầu ra:
  + KeyOut là tâm mới
  + ValOut danh sách các điểm

# Báo cáo kết quả thực nghiệm

## 4.1 Thu thập và tiền xử lý dữ liệu

Bài báo cáo này mang tính chất học thuật và nghiên cứu nên tập dữ liệu không được tác giả trực tiếp thu thập mà chỉ lấy từ nguồn trên Internet (Kaggle), đây là trang web chuyên cung cấp các tập dữ liệu dành cho việc nghiên cứu và ứng dụng các mô hình học máy.

- Có tổng cộng 2000 mẫu (vector) dữ liệu.

- Ma trận dữ liệu (X) gồm 7 cột:

+ Sex : Giới tính.

+ Marital status: Tình trạng hôn nhân.

+ Age : Tuổi.

+ Education: Trình độ học vấn.

+ Income: Thu nhập.

+ Occupation: Nghề nghiệp.

+ Settlement size: Kích thước thành phố nơi sinh sống

A screenshot of a table

Description automatically generated with low confidence

Hình ‑ Tập dữ liệu khách hàng chưa qua xử lý

* Với trường thông tin “ID” Không có vai trò gì trong bài toán nên sẽ thực hiện xóa bỏ thuộc tính này
* Trường “Sex”: các giá trị xuất hiện là 0 và 1 tương ứng với nam và nữ, thực hiện chuẩn hóa min-max
* Trường “Marital status”:
  + 0 tương ứng với “Độc thân”
  + 1 tương ứng với “Không độc thân” tức “Đã li hôn/Chia tay/Đã kết hôn/Góa phụ”

🡺 thực hiện chuẩn hóa min-max

* Trường “Age”: độ tuổi có giá trị từ 18-76, thực hiện chuẩn hóa min-max
* Trường “Education”:
  + 0 Tương ứng với “Các cấp bậc khác/không rõ”
  + 1 tương ứng với “Cấp bậc THPT”
  + 2 tương ứng với “Đại học/Cao đẳng”
  + 3 tương ứng với “Bậc đại học trở lên”

🡺 thực hiện chuẩn hóa min-max

* Trường “Income”: có giá trị số thực, thực hiện chuẩn hóa min-max
* Trường “Occupation”:
  + 0 tương ứng với: “Thất nghiệp/không có kỹ năng”
  + 1 tương ứng với “Công nhân/ nhân viên văn phòng”
  + 2 tương ứng với “Quản lý/tự kinh doanh”

🡺 thực hiện chuẩn hóa min-max

* Trường “Settlement size”:
  + 0 tương ứng với “Thành phố nhỏ”
  + 1 tương ứng với “Thành phố vừa”
  + 2 tương ứng với “Thành phố lớn”

🡺 thực hiện chuẩn hóa min-max

Tại sao lại phải chuẩn hóa min max trước khi thực hiện thuật toán Kmean???

**Trả lời:**

Chuẩn hóa min-max là một kỹ thuật chuẩn hóa dữ liệu phổ biến được sử dụng trước khi áp dụng các thuật toán machine learning như K-means clustering. Kỹ thuật này giúp tăng hiệu quả cho việc xử lý dữ liệu và đưa ra kết quả chính xác hơn.

Thuật toán K-means sử dụng khoảng cách Euclidean để tính toán khoảng cách giữa các điểm dữ liệu. Khi các giá trị của thuộc tính không được chuẩn hóa, các thuộc tính có giá trị lớn hơn sẽ ảnh hưởng nhiều hơn đến khoảng cách so với các thuộc tính có giá trị nhỏ hơn. Điều này gây ra sự thiếu chính xác và không tổng quát trong việc tính toán khoảng cách và phân loại.

Chuẩn hóa min-max có thể giúp giải quyết vấn đề này bằng cách đưa tất cả các giá trị thuộc tính về cùng một phạm vi giá trị, từ 0 đến 1 hoặc -1 đến 1. Khi đó, khoảng cách Euclidean giữa các điểm dữ liệu sẽ được tính toán dựa trên các giá trị đã được chuẩn hóa, giúp cho việc phân loại trở nên chính xác hơn.

Vì vậy, khi sử dụng thuật toán K-means để phân loại dữ liệu, chuẩn hóa min-max là một bước quan trọng trong việc chuẩn bị dữ liệu trước khi áp dụng thuật toán.

A picture containing text, screenshot, font, document

Description automatically generated

Hình ‑ Tập dữ liệu sau khi loại bỏ thuộc tính dư thừa và chuẩn hóa

## 4.2 Công cụ và thư viện sử dụng

1. Java Development Kit

Phiên bản sử dụng là java 1.8.0\_202

A black screen with white text

Description automatically generated with low confidence

Hình ‑ Phiên bản java

1. Apache Hadoop
2. Cấu hình file core-site.xml:

A picture containing text, font, screenshot

Description automatically generated

Hình ‑ Cấu hình file core-site.xml

Cấu hình này trong tệp core-site.xml đặt giá trị cho thuộc tính "fs.defaultFS" thành "hdfs://localhost:9000". Điều này có nghĩa là Hadoop sẽ sử dụng giao thức HDFS để truy cập vào tên miền "localhost" trên cổng 9000.

Giải thích chi tiết:

"fs.defaultFS" là một thuộc tính trong Hadoop, được sử dụng để chỉ định URL của hệ thống tập tin phân tán mặc định cho Hadoop. Nó xác định đường dẫn HDFS mà các ứng dụng MapReduce và Spark sẽ sử dụng để truy cập dữ liệu.

Trong trường hợp này, giá trị của "fs.defaultFS" được đặt thành "hdfs://localhost:9000", có nghĩa là Hadoop sẽ sử dụng giao thức HDFS để truy cập vào tên miền "localhost" trên cổng 9000. Điều này phù hợp với thiết lập cục bộ của Hadoop, nơi NameNode và DataNode được chạy trên cùng một máy chủ và sử dụng cổng 9000 để giao tiếp.

1. Cấu hình mapred-site.xml:

A picture containing text, screenshot, font

Description automatically generated

Hình ‑ cấu hình file Mapred-site.xml

File mapred-site.xml chứa các cấu hình cho MapReduce framework trong Hadoop. Trong đó, property mapreduce.framework.name được sử dụng để chỉ định tên của framework được sử dụng.

Trong trường hợp này, giá trị được cấu hình là yarn, tức là MapReduce framework sẽ chạy trên Apache Hadoop YARN (Yet Another Resource Negotiator). YARN là một phần mềm quản lý tài nguyên và lập lịch cho các ứng dụng phân tán, được sử dụng trong Hadoop để quản lý tài nguyên và lập lịch cho các ứng dụng MapReduce.

1. cấu hình hdfs-site.xml

A screenshot of a computer code

Description automatically generated with medium confidence

Hình ‑ Cấu hình file hdfs-site.xml

File hdfs-site.xml chứa các cấu hình cho Hadoop Distributed File System (HDFS), một phần quan trọng của Hadoop. Trong đó:

* Property *dfs.replication* được sử dụng để chỉ định số lượng bản sao của các block dữ liệu trong HDFS. Trong trường hợp này, giá trị được cấu hình là 1, có nghĩa là mỗi block dữ liệu trong HDFS chỉ có một bản sao.
* Property *dfs.namenode.name.dir* và *dfs.datanode.data.dir* được sử dụng để chỉ định vị trí thư mục lưu trữ dữ liệu cho NameNode và DataNode trong HDFS. Trong trường hợp này, thư mục **/hadoop-3.3.0/data/namenode** được sử dụng để lưu trữ metadata của HDFS bởi NameNode và thư mục /**hadoop-3.3.0/data/datanode** được sử dụng để lưu trữ dữ liệu của HDFS bởi DataNode.

1. yarn-site.xml

A picture containing text, screenshot, font

Description automatically generated

Hình ‑ cấu hình file yarn-site.xml

tệp cấu hình này định nghĩa hai thuộc tính:

* *yarn.nodemanager.aux-services* - Chỉ định danh sách các dịch vụ phụ được chạy trên NodeManager. Trong trường hợp này, chỉ có một dịch vụ phụ được cung cấp là "mapreduce\_shuffle".
* *yarn.nodemanager.auxservices.mapreduce.shuffle.class* - Chỉ định lớp xử lý ShuffleHandler sẽ được sử dụng cho "mapreduce\_shuffle" service.

Như vậy, thông qua tập tin cấu hình yarn-site.xml, NodeManager sẽ có thể biết được các dịch vụ phụ cần chạy và cách chúng được cấu hình.

## 4.3 Thực nghiệm và đánh giá kết quả

* Cài đặt mặc định mức ngưỡng (thress hold) : 0.01
* Số vòng lặp tối đa: 300
* Chỉ số đánh giá mô hình là Chỉ số DBI (Davies-Bouldin Index)

Thực hiện chạy chương trình với lần lượt số tâm cụm từ 3-10 để tìm ra số cụm tối ưu hàm đánh giá

* + K= 3

A black background with white text

Description automatically generated with low confidence

Hình ‑ Kết quả mô hình với k=3

* + K=4

A black background with white text

Description automatically generated with low confidence

Hình ‑ Kết quả mô hình với k=4

* + K=5

A black screen with white text

Description automatically generated with low confidence

Hình ‑ Kết quả mô hình với k=5

* + K=6

A black background with white text

Description automatically generated with low confidence

Hình ‑ Kết quả mô hình với k=6

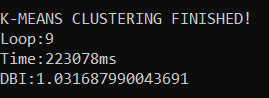
* + K=7

A black background with white text

Description automatically generated with low confidence

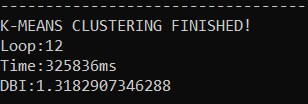
Hình ‑ Kết quả mô hình với k=7

* + K=8



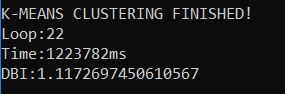
Hình ‑Kết quả mô hình với k=8

* + K=9



Hình ‑ Kết quả mô hình với k=9

* + K=10



Hình ‑Kết quả mô hình với k=10

🡺Như vậy có thể thấy với số lượng cụm bằng 8 sẽ cho ra mô hình có chỉ số đánh giá DBI là thấp nhất.Tuy nhiên sự chênh lệnh giữa các cụm không quá nhiều

Việc lựa chọn số cụm để phân chia không hoàn toàn phụ thuộc vào hệ số DBI mà còn phụ thuộc vào yêu cầu của bài toán, nghiệp vụ của doanh nghiệp đề ra cho mục đích thương mại.

# Tổng kết bài báo cáo

## Tổng Kết nội dung

Thông qua bài báo cáo này, chúng ta đã biết được lịch sử ra đời và phát triển của Hadoop và khả năng tính toán phân tán song song để áp dụng vào bài toán phân cụm trên tập dữ liệu lớn

## Hướng phát triển

Tuy nhiên bài toán phân cụm khách hàng sử dụng thuật toán Kmean và mô hình mapreduce được trình bày bên trên vẫn còn gặp một số hạn chế và chưa thực sự tối ưu:

* Single node Hadoop là một cài đặt Hadoop trên một máy tính đơn lẻ thay vì trên một cụm máy tính. Nó cho phép thực hành, phát triển và kiểm tra các ứng dụng Hadoop trên một môi trường thu nhỏ và không yêu cầu phải sử dụng nhiều máy tính.Với single node Hadoop, tất cả các thành phần Hadoop (bao gồm Namenode,Datanode, ResourceManager và NodeManager) được chạy trên một máy tính duy nhất. Điều này đơn giản hóa việc cấu hình và sử dụng Hadoop, và làm giảm đáng kể chi phí trong việc triển khai một cụm Hadoop thực sự. Tuy nhiên, single node Hadoop có giới hạn về khả năng xử lý dữ liệu, không phù hợp để xử lý các ứng dụng lớn hoặc có khối lượng dữ liệu lớn. Do vậy, khi triển khai ứng dụng Hadoop ta luôn thấy kết quả thời gian chạy vẫn còn rất lâu, ta cần sử dụng một cụm máy tính thực sự để tận dụng tối đa khả năng mở rộng của Hadoop.
* Với thuật toán Kmean, việc khởi tạo trọng tâm ban đầu cũng ảnh hưởng khá lớn tới thời gian và độ phức tạp của thuật toán. Để phát triển về sau chúng ta sẽ tìm hiểu và phát triển thuật toán Kmean++ intialization và K-Medoids Initialization
* Mô hình thực nghiệm trên chỉ đánh giá chất lượng mô hình dựa trên chỉ số Davies Bouldin Index (DBI) vì thế chưa thể đánh giá chính xác nhất về kết quả phân cụm của thuật toán. Chúng ta có thể phát triển thêm các đánh giá như: silhouette, Calinski-Harabasz,…

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | “TopDev - Việc làm IT hàng đầu,” [Trực tuyến]. Available: https://topdev.vn/blog/big-data/#big-data-la-gi. |
| [2] | "itnavi.com.vn," [Online]. Available: https://itnavi.com.vn/blog/mapreduce-nhung-uu-diem-va-cach-thuc-hoat-dong-cua-nen-tang-nay. |
| [3] | "Topdev.vn," [Online]. Available: https://topdev.vn/blog/hadoop-la-gi/. |
| [4] | "Wikipedia," [Online]. Available: https://vi.wikipedia.org/wiki/Apache\_Hadoop. |
| [5] | T. Q. Chiểu, Slide học tập môn Phân tích dữ liệu lớn. |
| [6] | "Hanghieugiatot.com," [Online]. Available: https://hanghieugiatot.com/silhouette-score-la-gi. |

PHỤ LỤC

1. Các câu lệnh trên Command Prompt :

* “Start-all.cmd” : thực hiện khởi động Hadoop
* “hdfs dfs -mkdir /Project\_input”: Tạo thư mục Project\_input để chứa các file dữ liệu
* “hadoop fs -put C:\kmeans\customer\_data.txt /Project\_input” : thực hiện đẩy file dữ liệu khách hàng “customer\_data.txt” lên thư mục Project\_input trong HDFS
* “hadoop jar C:\jar\KMeans.jar -Din /Project\_input/customer\_data.txt -Dlines 2000 -Dresult result\_project\_test.txt -Dmaxloop 300 -Dk 5 -Dthresh 0.01 -DNumReduceTask 1 -Dout /Project\_output”: Thực hiện chạy chương trình mapreduce với file code đã lập trình sẵn là Kmeans.jar với đường dẫn file input “/Project\_input/customer\_data.tx” , số điểm dữ liệu là 2000, kết quả sẽ được lưu vào file “result\_project\_test.txt” , số vòng lặp tối đa là 300, số cụm là 5, mức ngưỡng 0.01, số tác vụ Reduce là 1, thư mục lưu file kết quả là “Project\_output”.

1. Mã nguồn và dữ liệu

Xem tại đây: <https://github.com/MinhTuanCoder/KmeanWithMapreduce>