# TRƯỜNG ĐẠI HỌC CẦN THƠ
KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG


LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC

## *Đề tài*: NGHIÊN CỨU MÔ HÌNH MAPREDUCE VÀ NỀN TẢNG HADOOP, ỨNG DỤNG XÂY DỰNG HỆ THỐNG TÌM KIẾM TÀI LIỆU VĂN BẢN TIẾNG VIỆT

**Sinh viên thực hiện:**
**Huỳnh Phương Nam**
**MSSV: 1091686**

**Giáo viên hướng dẫn:**
**Ts. Ngô Bá Hùng**
**MSCB: 001124**

**Cần Thơ, 2013**

# TRƯỜNG ĐẠI HỌC CẦN THƠ
KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG


LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC

## ***Đề tài*: NGHIÊN CỨU MÔ HÌNH MAPREDUCE VÀ NỀN TẢNG HADOOP, ỨNG DỤNG XÂY DỰNG HỆ THỐNG TÌM KIẾM TÀI LIỆU VĂN BẢN TIẾNG VIỆT**

**Sinh viên thực hiện:**
**Huỳnh Phương Nam**
**MSSV: 1091686**

**Giáo viên hướng dẫn:**
**Ts. Ngô Bá Hùng**
**MSCB: 001124**

**Cán bộ phản biện:**
**Ks. Hà Duy An** **MSCB: 002366**
**Ks. Triệu Thanh Ngoan** **MSCB: 002481**

Luận văn được bảo vệ tại: Hội đồng chấm luận văn tốt nghiệp Bộ môn Mạng máy tính & Truyền thông, Khoa Công nghệ thông tin & Truyền thông, Trường Đại học Cần Thơ, vào ngày 07 tháng 5 năm 2013.

**Mã số đề tài:_____**

Có thể tìm hiểu luận văn tại:

- Thư viện Khoa CNTT & TT, Trường Đại học Cần Thơ.
- Website: http://www.cit.ctu.edu.vn/

# LỜI CẢM ƠN

Kính thưa !

Trong suốt quá trình học tập, nghiên cứu tại trường Đại học Cần Thơ, tôi đã nhận được sự hướng dẫn tận tình của quý Thầy Cô tại Trường, sự giúp đỡ, hỗ trợ hết lòng từ Gia đình, Bạn bè. Tôi xin được chân thành bày tỏ lòng biết ơn đến tất cả mọi người.

Tôi xin được gửi lời cảm ơn đến quý Thầy Cô trường Đại học Cần Thơ, những người đã tận tình giúp đỡ, hướng dẫn tôi trong quá trình thực hiện Luận văn tốt nghiệp cũng như suốt quãng thời gian tôi học tập, nghiên cứu tại trường Đại học Cần Thơ. Xin chân thành gửi lời cảm ơn đến thầy Ngô Bá Hùng vì những gì Thầy đã dành cho tôi.

Tôi xin được cảm ơn Gia đình, những người đã không ngừng giúp đỡ, động viên và hỗ trợ tôi trong suốt khoảng thời gian học tập xa nhà.

Cuối cùng, tôi xin được gửi lời cảm ơn đến bạn bè tôi, những người đã luôn sát cánh cùng tôi, vượt qua những gian nan thử thách. Cảm ơn bạn Nguyễn Thế Anh, người đã nhiệt tình hợp tác với tôi, cùng nhau thực hiện đề tài này.

Trân trọng !

Cần Thơ, ngày 29 tháng 4 năm 2013

Sinh viên thực hiện

**Huỳnh Phương Nam**

# MỤC LỤC

# KÍ HIỆU VÀ VIẾT TẮT

- TF - “Term Frequency”: Tần số xuất hiện của từ trong tài liệu.
- IDF - “Inverse Document Frequency”: Tần số nghịch đảo tài liệu của từ.
- TF-IDF - “Term Frequency-Inverse Document Frequency”: Trọng số của từ.
- HDFS - “Hadoop Distributed File System”: Hệ thống tập tin phân tán của Hadoop.

# TÓM TẮT

Chúng ta đang sống trong thời đại bùng nổ dữ liệu số. Mỗi ngày lượng dữ liệu lưu trữ không ngừng tăng lên và vượt ra khỏi khả năng xử lý của các hệ thống truyền thống. Đều đó đòi hỏi phải xây dựng một hệ thống mới, có khả năng xử lý được một lượng dữ liệu lớn, trong một khoảng thời gian chấp nhận được. Xuất phát từ nhu cầu đó, Hadoop đã ra đời dựa trên ý tưởng từ các bài báo về mô hình xử lý dữ liệu phân tán MapReduce của Google và Hệ thống tập tin của Google - Google File System (GFS). Hiện nay, Hadoop đã được ứng dụng rộng rãi trên khắp thế giới, tuy nhiên ở Việt Nam vẫn chưa có nhiều nghiên cứu ứng dụng công nghệ này.

Hadoop hoạt động trên ý tưởng của mô hình xử lý dữ liệu phân tán Mapreduce, cụ thể là hệ thống sẽ chia nhỏ nhiệm vụ lớn cho hàng loạt máy tính cùng nhau xử lý song song. Với ý tưởng đó, Hadoop có thể dễ dàng được triển khai trên các hệ thống máy tính có sẵn chứ không cần phải đầu tư những thiết bị phần cứng cao cấp, đắt tiền nào cả. Đặc biệt, Hadoop là một dự án mã nguồn mở và hoàn toàn miễn phí.

Ở Việt Nam những năm gần đây, lượng dữ liệu số tăng lên không ngừng, nhu cầu tìm kiếm tra cứu tài liệu văn bản tiếng Việt trên các kho dữ liệu lớn, cực lớn đã xuất hiện. Đề tài này nhằm nghiên cứu tổng quát mô hình MapReduce và nền tảng Hadoop, sau đó ứng dụng kết quả nghiên cứu vào việc xây dựng một công cụ tìm kiếm có xếp hạng kết quả dành cho tài liệu văn bản tiếng Việt.

# ABSTRACT

We're living in data explosion period. Everyday, amount storage of data continuous increases and over limits of processing of traditional systems. Therefore, building a new system is very necessary. That system can process amount of great data in an acceptable period. Starting from that demand, Hadoop was born based on ideas from articles about model of distributed data processing MapReduce of Google and Google File System (GFS). Nowadays, Hadoop has been widely applied all over the world. However, in Viet Nam, there has been little research in order to apply this technology.

Hadoop acts based on model of distributed data processing MapReduce, it means that it will divide small from big task for series of computer to parallel processing. With that idea, Hadoop can be easily deployed on computer systems available. It doesn't require investing of the high-end and expensive hardware. Especially, Hadoop is a opent-source project and extremely free

In Vietnam, recent years, amount of data continuous increases, demand for searching text document by Vietnamese on great storage of data, extra- great has appeared. The objective of this topic was study general MapReduce model and Hadoop basis, then applying the results of research into building a search engine ranking results for Vietnamese documents.

## TỪ KHÓA

- MapReduce
- Hadoop
- HDFS
- Vector space model
- TF-IDF

# CHƯƠNG 1: TỔNG QUAN

## 1.1. ĐẶT VẤN ĐỀ

Với sự phát triển nhanh chóng, hiện nay công nghệ thông tin đã thật sự lấn sâu vào mọi lĩnh vực trong đời sống, máy tính xuất hiện ở khắp mọi nơi, dữ liệu của mọi lĩnh vực dần được số hóa để thuận tiện trong việc lưu trữ, phân phối, tra cứu, v.v... Điều này đã dẫn đến sự bùng nổ của dữ liệu số. Hiện tại, nhiều tổ chức đang gặp rất nhiều khó khăn khi phải thao tác với kho dữ liệu số quá lớn của mình, các hệ thống xử lý truyền thống bị quá tải, đặt ra yêu cầu phải xây dựng hệ thống xử lý mới, có khả năng làm việc hiệu quả với dữ liệu lớn.

Ở Việt Nam, một trong những nhu cầu thao tác với dữ liệu lớn được đặt ra là tìm kiếm tài liệu văn bản tiếng Việt, phục vụ cho nhu cầu học tập, nghiên cứu, v.v. Việc tìm kiếm chẳng những đòi hỏi tính chính xác cao, mà còn cần được thực hiện trong một khoảng thời gian chấp nhận được. Đây là một vấn đề đáng được quan tâm và nghiên cứu giải quyết.

## 1.2. LỊCH SỬ GIẢI QUYẾT VẤN ĐỀ

Tại trường Đại học Cần Thơ, vấn đề xây dựng một công cụ tìm kiếm cho tài liệu văn bản tiếng Việt đã từng được nghiên cứu trước đây trong các đề tài luận văn cao học:

- Đề tài: "Nghiên cứu nền tảng tính toán song song với MapReduce và Hadoop. Áp dụng cho việc xây dựng wordnet tiếng Việt và tạo chỉ mục tài liệu"; tác giả Nguyễn Minh Thuận, Nguyễn Trọng Thức; năm 2012 [4].

- Đề tài: "Xây dựng một ứng dụng minh họa cho khả năng của MongdoDB"; tác giả Bùi Thị Hồng Phúc; năm 2012 [5].

Các tác giả đã dùng mô hình tính toán phân tán MapReduce được cài đặt trên nền tảng Hadoop, kết hợp với Hệ quản trị cơ sở dữ liệu MongoDB để giải quyết vấn đề. Công cụ đã cơ bản giải quyết được vấn đề, tuy nhiên vẫn còn một số hạn chế: Việc xử lý yêu cầu tìm kiếm còn chậm, không thể bổ sung tài liệu mới vào kho tài liệu, chương trình hỗ trợ tìm kiếm là ứng dụng desktop, khó triển khai thành công cụ

tìm kiếm trực tuyến.

## 1.3. MỤC TIÊU VÀ PHẠM VI NGHIÊN CỨU

Sau khi phân tích yêu cầu của đề tài và bàn bạc với thành viên còn lại của nhóm, trong khuôn khổ luận văn này, mục tiêu và phạm vi nghiên cứu của tôi là:

- Tìm hiểu nguyên tắc hoạt động của mô hình xử lý dữ liệu phân tán MapReduce, cách thức cài đặt và sử dụng nền tảng Hadoop.

- Vận dụng mô hình MapReduce và nền tảng Hadoop, xây dựng hệ thống xử lý yêu cầu tìm kiếm trên tập tài liệu văn bản tiếng Việt có kích thước lớn, đã được xử lý lập chỉ mục trước đó. Kết quả tìm kiếm có sự sắp xếp dựa theo mức độ liên quan giữa biểu thức tìm kiếm và nội dung của mỗi tài liệu. Hệ thống được xây dựng dưới dạng một ứng dụng trực tuyến, có thời gian xử lý cho mỗi yêu cầu tìm kiếm đủ nhanh, để người dùng chấp nhận được.

- Triển khai hệ thống trên một mạng các máy tính được kết nối, kiểm tra đánh giá khả năng hoạt động của hệ thống.

## 1.4. PHƯƠNG PHÁP THỰC HIỆN

Để hoàn thành nội dung được phân công, trước tiên cần tìm kiếm, nghiên cứu các tài liệu về:

- Mô hình xử lý phân tán MapReduce.

- Cách thức cài đặt và khai thác nền tảng Hadoop.

- Mô hình không gian vector, tính độ tương đồng giữa các tài liệu văn bản.

Sau khi nắm được cơ sở lý thuyết, sẽ phân tích thiết kế giải thuật cho từng thành phần của hệ thống, lựa chọn ngôn ngữ lập trình, tiến hành cài đặt các giải thuật, và tiếp tục nghiên cứu các vấn đề phát sinh.

Cuối cùng sẽ kết hợp với thành viên còn lại của nhóm, tổng hợp các hệ thống

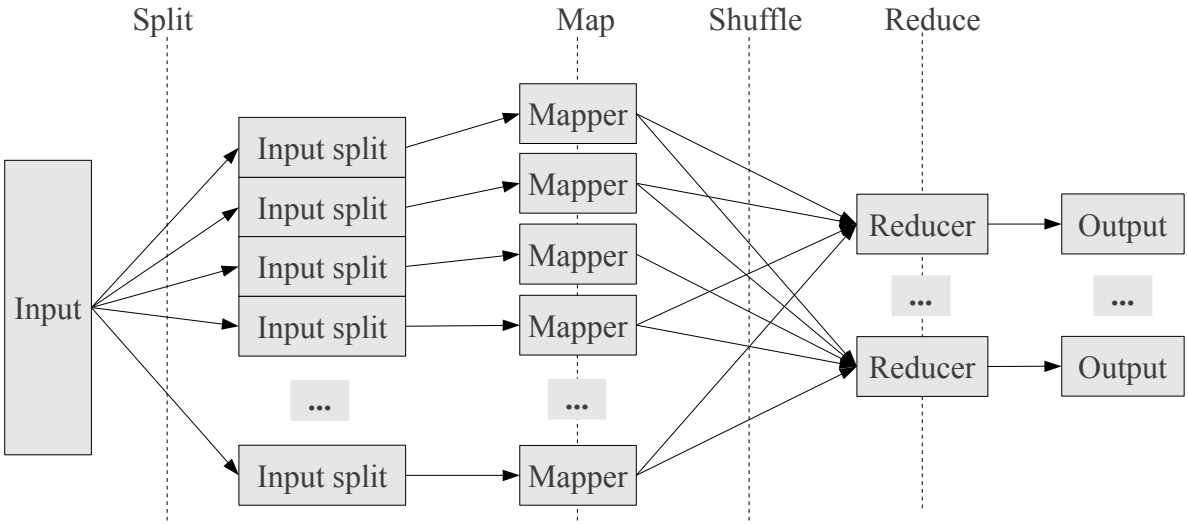
do mỗi người thực hiện lại thành một hệ thống hoàn chỉnh, sau đó tiến hành triển khai, kiểm tra đánh giá khả năng hoạt động.

# CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

## 2.1. MÔ HÌNH XỬ LÝ DỮ LIỆU PHÂN TÁN MAPREDUCE

### 2.1.1. Giới thiệu chung

Năm 2004, Google công bố mô hình xử lý dữ liệu phân tán MapReduce, Mô hình này là sáng kiến của một nhóm các kỹ sư Google, khi nghiên cứu tìm kiếm giải pháp mở rộng cỗ máy tìm kiếm của họ. Có thể coi MapReduce là một mô hình lập trình, hay một giải thuật lập trình, chuyên dùng để giải quyết vấn đề về xử lý dữ liệu lớn. Mô hình này cơ bản gồm hai thao tác chính là Map và Reduce, với ý tưởng là chia công việc lớn ra thành nhiều công việc nhỏ, giao cho nhiều máy tính cùng thực hiện - thao tác Map, sau đó tổng hợp kết quả lại - thao tác Reduce.




Hình 2.1.1.a: Mô hình tổng quát của MapReduce

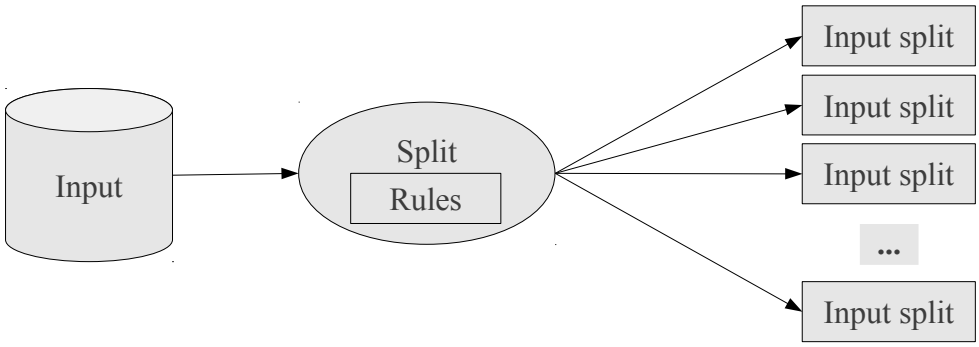
Trong mô hình trên, ngoài hai quá trình cơ bản là Map và Reduce đã được trình bày, còn có thêm hai quá trình nữa là Split và Shuffle, hai quá trình này lần lượt giữ vai trò: phân chia dữ liệu đầu vào, tạo tiền đề cho quá trình Map và gom nhóm dữ liệu đầu ra của quá trình Map, tạo tiền đề cho quá trình Reduce.

MapReduce định nghĩa dữ liệu dưới dạng các cặp <key, value> - <khóa, giá trị>; ví dụ, key có thể là tên của tập tin và value nội dung của tập tin, hoặc key là địa chỉ URL và value là nội dung tại URL, v.v. Dữ liệu được định nghĩa theo dạng này

linh hoạt hơn các bảng dữ liệu quan hệ hai chiều truyền thống (quan hệ cha - con hay còn gọi là khóa chính - khóa phụ).

### 2.1.2. Quá trình Split

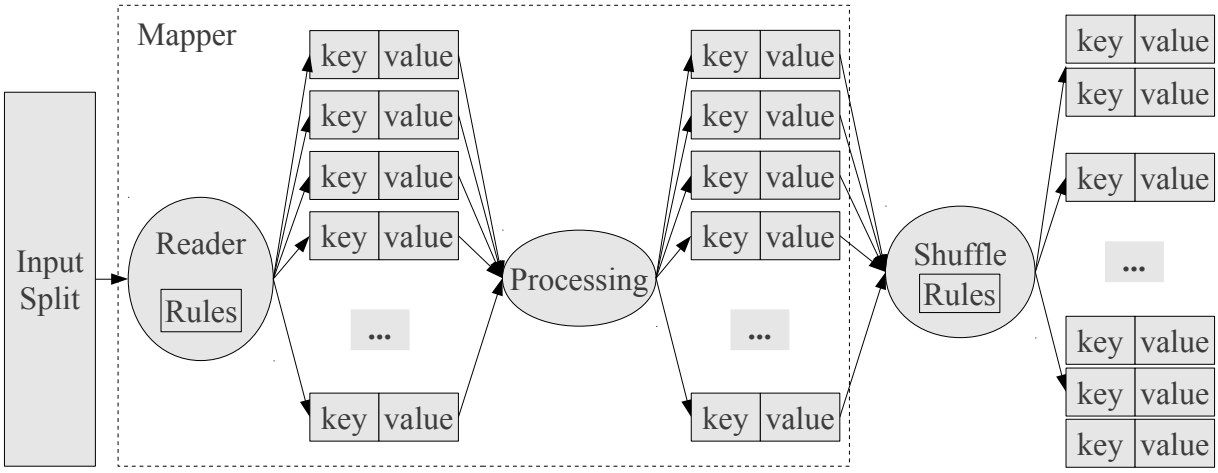
Để có thể phân tán công việc trên hệ thống máy tính, trước tiên cần phải phân nhỏ khối dữ liệu đầu vào cần xử lý ra thành nhiều phần, rồi sau đó mới có thể phân công cho mỗi máy xử lý một phần trong số đó. Quá trình phân chia dữ liệu này được gọi là Split, Split sẽ dựa vào một bộ tiêu chí được đặt ra trước để chia nhỏ dữ liệu, mỗi mảnh dữ liệu được chia nhỏ như vậy gọi là một input split.




Hình 2.1.2.a - Quá trình Split

### 2.1.3. Quá trình Map và Shuffle

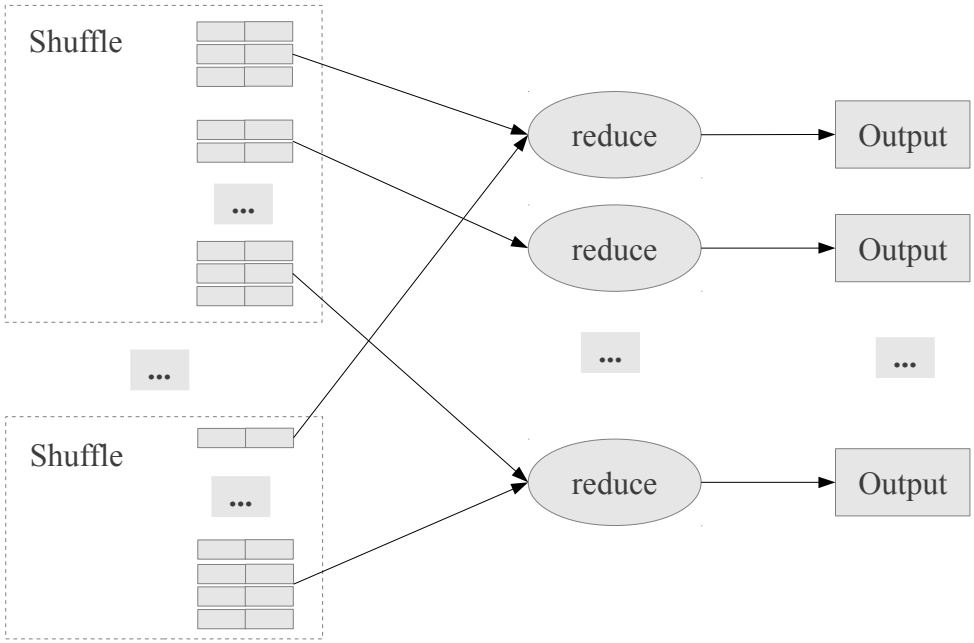
Sau khi các input split được tạo ra, Quá trình Map được thực hiện - hệ thống sẽ phân bố các input split về các máy xử lý, các máy được phân công sẽ tiếp nhận và xử lý input split được giao, ta gọi quá trình diễn ra trên nội bộ mỗi máy trong quá trình Map là Mapper. Trước khi được xử lý, input split được định dạng lại thành dữ liệu chuẩn của MapReduce - dữ liệu có dạng các cặp <key, value>. Kết thúc quá trình Mapper trên mỗi máy, dữ liệu đầu ra cũng có dạng các cặp <key, value>, chúng sẽ được chuyển sang cho quá trình Shuffle để phân nhóm theo tiêu chí đã được định trước, chuẩn bị cho bước xử lý phân tán tiếp theo. Như vậy, quá trình Shuffle sẽ được thực hiện một cách nội bộ trên mỗi máy chạy Mapper.




Hình 2.1.3.a - Quá trình Mapper và Shuffle trên một máy.

### 2.1.4. Quá trình Reduce

Quá trình Shuffle diễn ra trên nhiều máy nhưng do sử dụng chung một tiêu chí đã được định trước, nên việc phân nhóm dữ liệu trên các máy có sự thống nhất. Các nhóm dữ liệu tương ứng với nhau trên tất cả các máy chạy Shuffle sẽ được gom lại chuyển về cho cùng một máy xử lý, cho ra kết quả cuối cùng. Toàn bộ quá trình này được gọi là Reduce, quá trình xử lý trên từng máy trong quá trình Reduce là quá trình Reducer.




Hình 2.1.4.a: Quá trình Reduce

### 2.1.5. Một số bài toán ứng dụng mô hình MapReduce

**Grep phân tán**

Grep là một tiện ích dòng lệnh dùng cho việc tìm kiếm trên tập dữ liệu văn bản. Khi áp dụng mô hình MapReduce, trong quá trình Map, mỗi Mapper sẽ làm việc với một tập con của tập dữ liệu văn bản, công việc của mỗi Mapper là tìm kiếm và đánh dấu những dòng khớp với biểu thức tìm kiếm trong tập dữ liệu văn bản mà mình phụ trách. Kết quả của các Mapper sẽ được quá trình Reduce gom lại tạo thành kết quả cuối cùng.

**Sắp xếp phân tán**

Mô hình MapReduce rất phù hợp với bài toán sắp xếp dữ liệu. Trong quá trình Map, mỗi Mapper sẽ chỉ giữ nhiệm vụ đọc dữ liệu lên, Shuffle sẽ phân nhóm dữ liệu theo từng khoảng giá trị, Quá trình Reduce sẽ chịu trách nhiệm sắp xếp dữ liệu, mỗi Reducer sẽ sắp xếp dữ liệu trên khoảng giá trị được phân công.

Một ví dụ cụ thể về việc sắp xếp dữ liệu: Bài toán sắp xếp một tập dữ liệu nhiệt độ. Thay vì sắp xếp toàn bộ dữ liệu bằng một chương trình tuần tự, ta có thể xử lý bằng mô hình MapReduce như sau: Tại quá trình Map, dữ liệu sẽ được đọc lên và định dạng thành các cặp <ngày đo, nhiệt độ>. Tại quá trình Shuffle, ta sẽ phân chia dữ liệu theo từng khoảng giá trị của trường "nhiệt độ", trước khi chuyển qua cho Reduce sắp xếp. Như vậy, mỗi Reducer sẽ chỉ sắp xếp những dữ liệu nằm trong một khoảng nhất định. Ta dễ dàng tổng hợp kết quả sắp xếp của các Recducer, để tạo ra kết quả sắp xếp toàn cục.

## 2.2. HADOOP - NỀN TẢNG LẬP TRÌNH THEO MÔ HÌNH MAPREDUCE

### 2.2.1. Giới thiệu chung

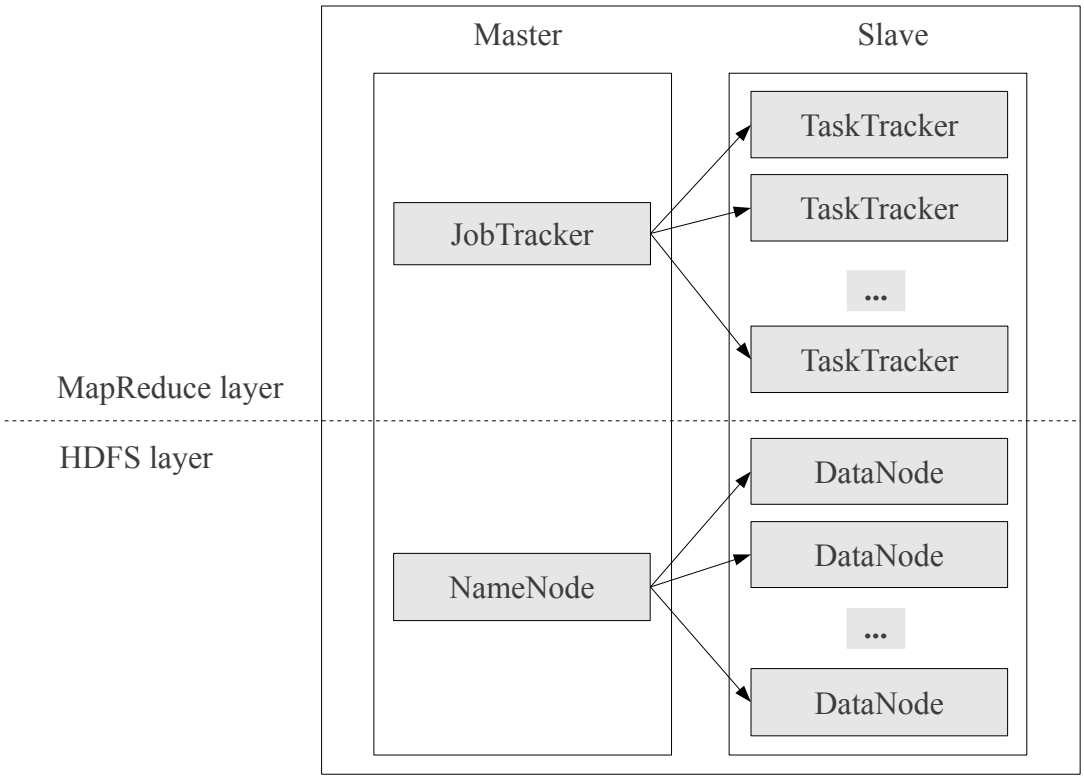
Hadoop là một nền tảng nguồn mở, được Dough Cutting tạo ra khi ông nghiên cứu về Nutch - một ứng dụng tìm kiếm. Hadoop được viết bằng Java, dùng hỗ trợ xây dựng, thực thi các ứng dụng tính toán phân tán theo mô hình MapReduce. Hadoop cluster là hệ thống máy tính đã được triển khai nền tảng Hadoop, một Hadoop cluster bao gồm hai thành phần cơ bản là kiến trúc MapReduce và hệ thống tập tin phân tán

HDFS, Trong đó:

- Kiến trúc MapReducer gồm hai phần: TaskTracker - trực tiếp thực thi các tác vụ xử lý dữ liệu, JobTracker - quản lý và phân chia công việc cho các TaskTracker.

- Hệ thống HDFS gồm hai phần: DataNode - nơi trực tiếp lưu trữ dữ liệu, mỗi DataNode chịu trách nhiệm lưu trữ một phần dữ liệu của hệ thống, NameNode - quản lý các DataNode, dẫn đường cho các yêu cầu truy xuất dữ liệu.

Kiến trúc của Hadoop cluster là kiến trúc Master-Slave, và cả hai thành phần MapReduce và HDFS đều tuân theo kiến trúc này .




Hình 2.2.1.a: Các thành phần của Hadoop cluster

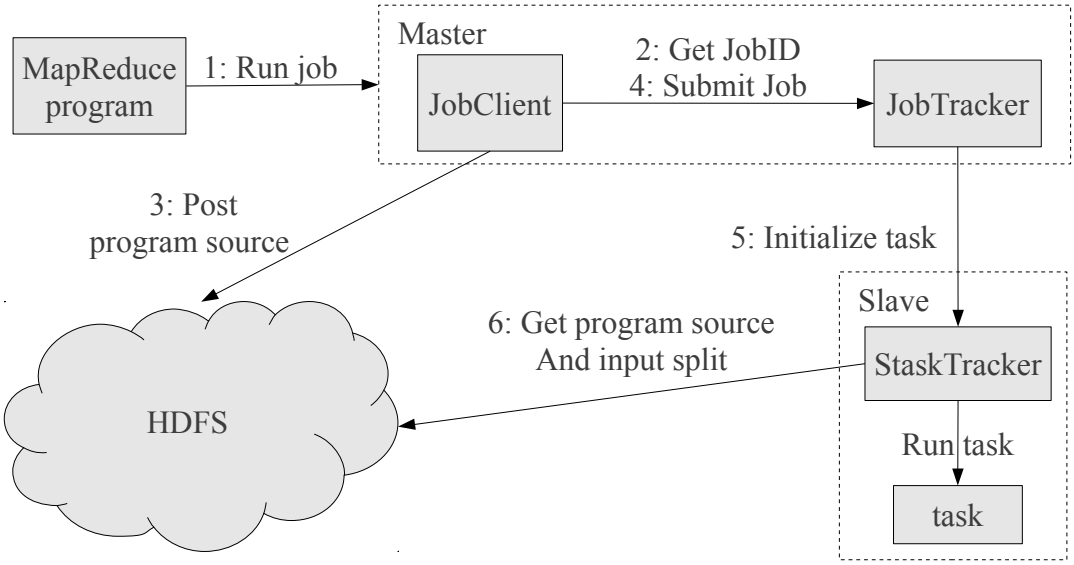
### 2.2.2. MapReduce Layer

**JobTracker và TaskTracker**

Trong Hadoop, mỗi quá trình xử lý MapReduce được gọi là một job. Việc thực hiện job sẽ được quản lý bởi hai đối tượng là JobTracker và TaksTracker. JobTracker hoạt động tại máy master có nhiệm vụ quản lý toàn bộ hệ thống gồm việc tạo và quản lý job, phân bố dữ liệu và phân công công việc cho các TaskTracker, xử lý lỗi, v.v.Tại mỗi máy slave có một TaskTracker hoạt động để tạo các task xử lý theo yêu cầu của JobTracker. Ngoài ra, định kỳ mỗi khoảng thời gian, TaskTracker phải gửi tín hiệu HeartBeat về JobTracker để thông báo rằng nó vẫn đang còn hoạt động. Điều này đảm bảo JobTracker lập thời biểu công việc chính xác và hiệu quả cho cả hệ thống.

**Cơ chế hoạt động mô hình MapReduce trong Hadoop**

Mỗi khi có yêu cầu thực thi một ứng dụng MapReduce, JobTracker sẽ tạo ra một JobClient và chép toàn bộ code thực thi cần thiết của job đó lên hệ thống tập tin phân tán HDFS, mỗi JobClient sẽ được gán một jobID duy nhất. Tiếp theo JobClient sẽ gửi một yêu cầu thực thi job lên JobTracker, JobTracker dựa theo yêu cầu của JobClient, sẽ gửi yêu cầu khởi tạo task kèm theo các thông tin phân công công việc đến các TaskTracker. Mỗi TaskTracker sẽ dựa vào thông tin phân công lần lượt thực hiện: Khởi tạo map task hoặc reduce task, chép toàn bộ code thực thi trên HDFS về, thực hiện công việc được phân công. Sau khi thực hiện xong, TaskTracker sẽ thông báo cho JobTracker và tự giải phóng.




Hình 2.2.2.a: Cơ chế hoạt động của JobTracker và TaskTracker trong Hadoop

### 2.2.3. Hadoop Distributed File System Layer

**Giới thiệu chung**

Hadoop Distributed File System (HDFS) là một hệ thống tập tin phân tán, được thiết kế để chạy trên hệ thống nhiều máy tính được nối mạng với nhau, có khả năng chịu lỗi cao và có thể triển khai trên hệ thống phần cứng không đòi hỏi cấu hình đắt tiền. Có rất nhiều đặc điểm giống nhau giữa HDFS và những hệ thống tập tin phân tán khác. Tuy nhiên, HDFS có những đặc điểm nổi bật riêng giúp nó có khả năng hỗ trợ tốt cho các ứng dụng xử lý dữ liệu lớn.

**Những đặc điểm của HDFS**

Dữ liệu lưu trữ cực lớn: HDFS được thiết kế để lưu trữ những tập tin với kích thước hàng trăm megabyte, gigabyte hay terabyte. Ngày nay, hệ thống HDFS có thể lưu trữ lên đến petabyte dữ liệu.

Xử lý lỗi phần cứng: HDFS không đòi hỏi phần cứng cấu hình cao đối với hệ thống máy tính. Vì vậy, việc xảy ra lỗi trên các thiết bị phần cứng là hoàn toàn có thể xảy ra một cách thường xuyên. Một hệ thống HDFS có thể bao gồm hàng ngàn máy xử lý, mỗi máy (node) lưu trữ một phần của dữ liệu. HDFS có cơ chế quản lý toàn bộ các node đang chạy trên hệ thống để nhận biết node nào đang rảnh và node nào bị lỗi. Những công việc và dữ liệu được xử lý tại node bị lỗi đó sẽ được chuyển sang node rảnh của hệ thống để xử lý lại.

Dữ liệu chặt chẽ: HDFS hoạt động theo cơ chế ghi một lần - đọc nhiều lần. Mỗi tập tin sẽ được tạo, ghi dữ liệu và đóng lại hoàn toàn. Việc cập nhật ghi thêm dữ liệu vào tập tin là không thể thực hiện trên HDFS. Dữ liệu có thể được truy xuất nhiều lần nhưng vẫn đảm bảo tính nhất quán. Cơ chế thích hợp cho những ứng dụng đọc dữ liệu theo dạng tuần tự.

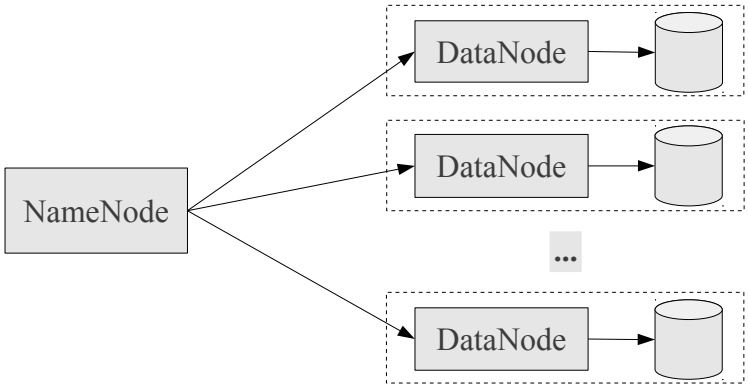
Di chuyển tính toán thay vì di chuyển dữ liệu: những yêu cầu tính toán của ứng dụng sẽ được thực hiện tại node chứa dữ liệu gần nhất với nó. Điều này càng hiệu quả đối với dữ liệu lớn và hệ thống mạng băng thông hẹp. HDFS cung cấp giao diện cho ứng dụng tìm kiếm và di chuyển chính nó đến vị trí dữ liệu gần nhất.

Chạy trên nhiều nền tảng và thiết bị: HDFS được thiết kế để dễ dàng di chuyển từ nền tảng này sang nền tảng khác, thiết bị này sang thiết bị khác. Điều này tạo điều kiện thuận lợi cho việc ứng dụng HDFS một cách rộng rãi.

**Các khái niện trên HDFS**

Block: Mỗi đĩa cứng có một kích thước block nhất định. Đó là kích thước dữ liệu nhỏ nhất có thể được ghi và đọc trên đó. Kích thước block cho những tập tin hệ thống cho những đĩa lưu trữ đơn thường khoảng vài kilobyte. Việc này giúp người dùng dễ dàng đọc hoặc ghi tập tin với chiều dài đó. HDFS cũng có quy định về kích thước block. Mặc định mỗi block trên HDFS có kích thước là 64MB.

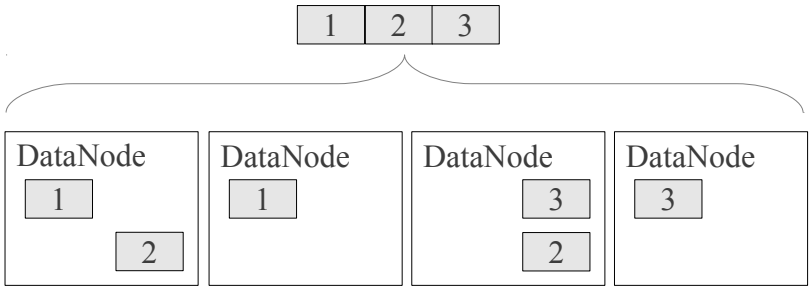
NameNode và DataNode: HDFS là một kiến trúc Master/Slave. HDFS cluster là hệ thống máy tính đã được triển khai HDFS. Trong một cluster HDFS có duy nhất một NameNode đóng vai trò giống như một master, dùng quản lý không gian tên của hệ thống tập tin (file system namespace) và điều phối việc truy xuất dữ liệu. Ngoài ra, còn có các DataNode đóng vai trò như các slave. Mỗi DataNode chịu trách nhiệm quản lý thông tin lưu trữ của các dữ liệu trên máy mà nó đang chạy. HDFS cung cấp một không gian tên cho phép dữ liệu của người dùng lưu trữ trong các tập tin. Mỗi tập tin sẽ được tách thành một hoặc nhiều block được lưu trữ trong một tập hợp những DataNode. Dựa vào không gian tên, NameNode có thể thực hiện các thao tác như đóng, mở và đổi tên tập tin và thư mục. NameNode cũng xác định sơ đồ lưu trữ các block của DataNode. Ngoài nhiệm vụ đáp ứng yêu cầu đọc, ghi dữ liệu do Namenode chuyển đến, các DataNode cũng có nhiệm vụ tạo, xóa và nhân rộng các blocks theo những chỉ thị từ NameNode.




Hình 2.2.3.a: Kiến trục Master/Slave của hệ thống tập tin phân tán Hadoop (HDFS)

File System Namespace: HDFS tổ chức tập tin phân cấp theo mô hình truyền thống. Với HDFS người dùng cũng có thể tạo, xóa, di chuyển, đổi tên tập tin thư mục như các hệ thống tập tin truyền thống thông thường.

Data Replication: HDFS được thiết kế để lưu trữ các tập tin cực lớn. Trên một Hadoop cluster mỗi tập tin được chia thành nhiều block có thứ tự và lưu trữ trên nhiều máy. Việc nhân bản các block nhằm tăng khả năng chịu lỗi cho hệ thống. Mỗi block sẽ được nhân bản bao nhiêu lần tùy theo cấu hình của hệ thống. Các DataNode có nhiệm vụ lưu trữ các block mà nó được NameNode phân công.




Hình 2.2.3.b: Nhân bản block trong HDFS

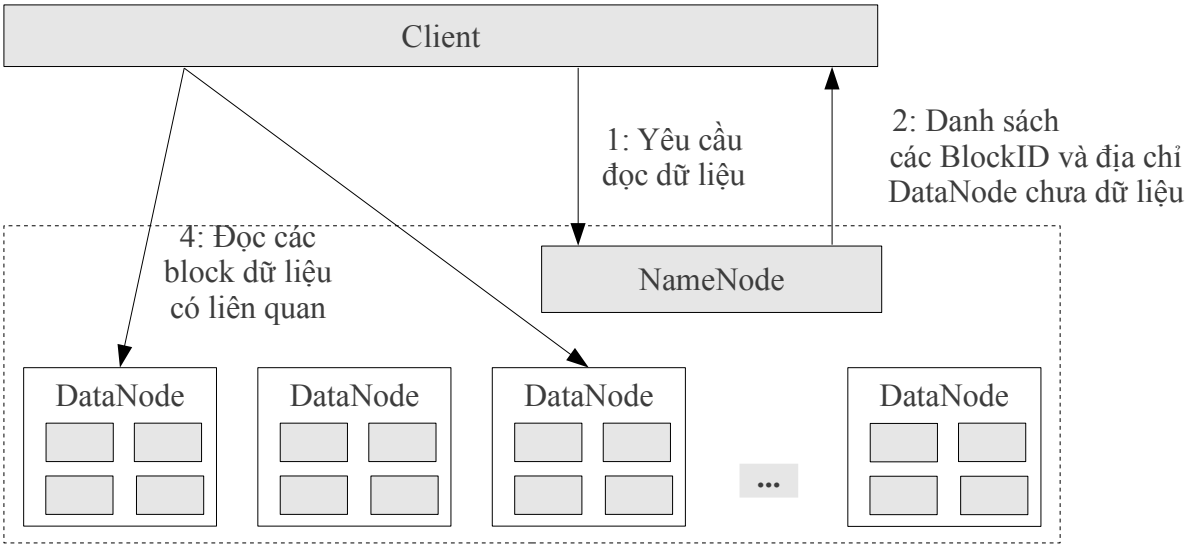
**Lỗi đĩa, thông điệp HeartBeat và nhân bản lại các block**

Định kỳ, mỗi DataNode sẽ gởi đến NameNode một thông điệp gọi là HeartBeats để xác định tình trạng hoạt động của DataNode. Nếu sau một khoảng thời gian quy định mà không thấy DataNode gởi HeartBeats, NameNode sẽ đánh dấu là DataNode đó bị lỗi và không ra bất kỳ thao tác nào cho nó nữa. Bất kỳ dữ liệu gì do DataNode bị lỗi quản lý đều xem như không còn nữa. NameNode sẽ tìm một bản sao khác của các block trên DataNode bị lỗi và sao chép toàn bộ công việc của DataNode bị lỗi cùng bản sao của các block cho DataNode nào còn rảnh để thực hiện tiếp công việc. Khi số lượng bản sao của một block nhỏ hơn giá trị được chỉ định trước, NameNode sẽ khởi tạo quá trình nhân bản bất cứ khi nào có thể.

**Truy xuất dữ liệu trên HDFS**

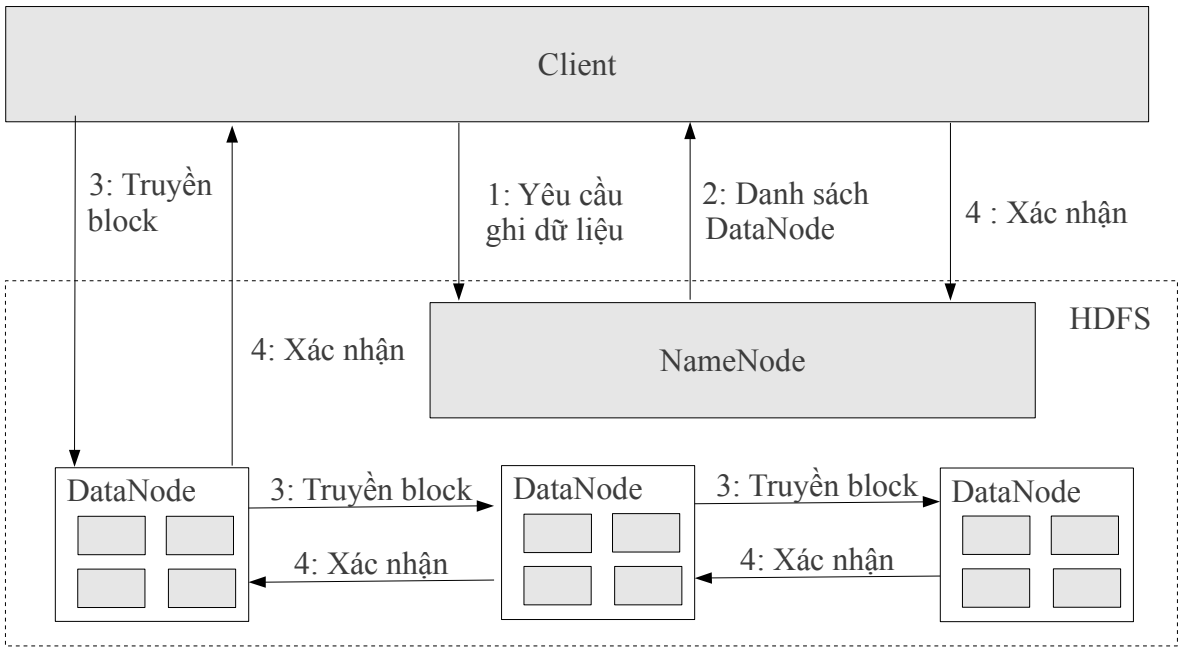
Đọc dữ liệu trên HDFS: Khi chương trình client gởi một yêu cầu đọc dữ liệu tới HDFS, hệ thống sẽ gởi một yêu cầu đến NameNode để hỏi về vị trí các block có liên quan đến dữ liện cần đọc. Sau khi có được danh sách vị trí các block có liên quan,

Client sẽ kết nối trực tiếp đến DataNode để đọc dữ liệu.




Hình 2.2.3.c: Quá trình đọc dữ liệu trên HDFS

Ghi dữ liệu trên HDFS: Khi chương trình client có tập tin dữ liệu cần ghi lên HDFS, đầu tiên tập tin dữ liệu phải được ghi vào bộ nhớ cục bộ của máy tính chạy chương trình client. Khi tập tin cục bộ đã tích lũy đủ dung lượng của một block, hoặc quá trình ghi tập tin cục bộ đã hoàn toàn kết thúc, chương trình client sẽ nhận về một danh sách những DataNode được NameNode chỉ định sẽ chứa tập tin dữ liệu này. Sau đó, chương trình client sẽ gởi bản sao của tập tin cần ghi đến DataNode đầu tiên. DataNode đầu tiên sẽ trích lấy một phần của tập tin để ghi xuống bộ nhớ của mình, sau đó chuyển toàn bộ phần còn lại cho DataNode khác, DataNode khác sẽ thực hiện lại công việc mà DataNode đầu tiên đã làm, cứ tiếp tục như vậy cho đến khi toàn bộ nội dung của tập tin được ghi hết. Có thể nói, việc ghi dữ liệu trên hệ thống HDFS được thực hiện theo cơ chế ống dẫn.




Hình 2.2.3.d: Quá trình ghi dữ liệu trên HDFS theo cơ chế ống dẫn

**SecondaryNameNode**

Trong HDFS cluster có duy nhất một NameNode, hệ thống không thể hoạt động được nếu như không có NameNode. Vì tính chất quan trọng đó, việc sao lưu dự phòng cho NameNode là rất cần thiết. Đó chính là nhiệm vụ của Secondary NameNode. Định kỳ sau mỗi khoảng thời gian, Secondary NameNode sẽ kết nối đến NameNode để cập nhật các tập tin về cấu hình, trạng thái của hệ thống, toàn bộ các tập tin này sẽ được lưu lại thành một CheckPoint, dùng cho việc khôi phục lại NameNode nếu NameNode bị lỗi. Đồng thời, NameNode là một dịch vụ phải phục vụ rất nhiều DataNode trong hệ thống. Secondary NameNode còn có khả năng chia tải với NameNode. Ngoài ra, việc định kỳ tạo CheckPoint giúp hệ thống chuyển sang trạng thái hoạt động nhanh hơn.

### 2.2.4. Kiểm tra tình trạng hoạt động của hệ thống Hadoop

Ta có thể kiểm tra tình trạng hoạt động của các MapReduce job thông qua trình duyệt web, với địa chỉ http:<JobTracker>:50030, giá trị <JobTracker> chính là IP hoặc tên miền của máy đang chạy JobTrakcer, tức máy master của hệ thống. Nhờ đó, ta có thể biết được MapReduce job đã hoàn tất, bị lỗi hay đang chạy, chạy được bao nhiêu

phần, node nào đang chạy các TaskTracker, bao nhiêu tác vụ Reducer, bao nhiêu tác vụ Mapper đang chạy trên các node, số phần trăm kết quả đạt được của các tác vụ, mỗi máy đã xử lý bao nhiêu input split, v.v.

Tương tự với HDFS, ta có thể sử dụng trình duyệt web, truy cập tới địa chỉ http:<NameNode>:50070, với <NameNode> là địa chỉ IP, hoặc tên miền của máy chạy NameNode, để biết được các thông tin về trạng thái hoạt động của hệ thống.

## 2.3. XÂY DỰNG MỘT CHƯƠNG TRÌNH CHẠY TRÊN NỀN HADOOP

### 2.3.1. Các lớp cơ bản trong một chương trình Hadoop

**Các kiểu dữ liệu cơ bản**

Để xây dựng chương trình, Hadoop xây dựng gói org.apache.hadoop.io hỗ trợ các kiểu dữ liệu phù hợp với Hadoop cho Java, gồm có:

- NullWritale: tương ứng với kiểu dữ liệu Null trong Java.
- Text: tương ứng với kiểu dữ liệu String trong Java.
- BytesWritable: tương ứng với kiểu Byte trong Java.
- BooleanWritable: tương ứng với kiểu Boolean trong Java.
- IntWritable: tương ứng với kiểu Integer trong Java.
- LongWritable: tương ứng với kiểu Long trong Java.
- FloatWritable: tương ứng với kiểu Float trong Java.
- DoubleWritable: tương ứng với kiểu Double trong Java

**Lớp Mapper**

Đây là lớp hỗ trợ thực hiện quá trình Map trong hệ thống. Lập trình viên sẽ viết

một lớp mới, thừa kế lại lớp Mapper. Có thể định nghĩa lại các phương thức trong lớp Mapper cho phù hợp, có hai phương thức qua trọng cần phải quan tâm là:

- Phương thức run(): Lập trình viên có thể định nghĩa lại phương thức này để kiểm soát việc đọc và phân phát dữ liệu từ input split.

- Phương thức map(): Đây là phương thức quan trọng nhất, trong hầu hết các trường hợp Lập trình viên phải định nghĩa lại phương thức này, phương thức này được thiết kế để mỗi lần nhận vào và xử lý một cặp <key, value>.

```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
                String line = value.toString();
                StringTokenizer tokenizer = new StringTokenizer(line);
                while (tokenizer.hasMoreTokens()) {
                        word.set(tokenizer.nextToken());
                        output.collect(word, one);
                }
        }
}
```

Ví dụ về sử dụng lớp Mapper trong chương trình Word count

**Lớp Partitioner**

Sử dụng lớp Partitioner giúp chúng ta có thể tùy biến, phân nhóm các cặp <key, value> đầu ra của quá trình Mapper trên mỗi map task. Nếu không sử dụng lớp này trong chương trình MapReduce, dữ liệu đầu ra của quá trình Mapper sẽ được gom lại thành một nhóm duy nhất.

```
public static final class SortReducerByValuesPartitioner implements Partitioner {
        public int getPartition(Text key, Text value, int numPartitions) {
                return key.toString().charAt(0) % numPartitions;
        }

        public void configure(JobConf conf) {
        }
}
```

Ví dụ về sử dụng lớp Partitioner để phân nhóm dữ liệu theo thuộc tính key

**Lớp hỗ trợ Combiner**

Combiner có thể được hoặc không được sử dụng trong chương trình MapReduce, mục đích của tác vụ này là giảm lượng dữ liệu gởi đi từ các map task tới các reduce task. Bản chất của tác vụ Combiner là thực hiện tác vụ Reducer tại từng map task trước khi gửi đi thực hiện Reducer một lần nữa tại reduce task. Mỗi map task sẽ thực hiện một hoặc nhiều tác vụ Combiner, mỗi Combiner sẽ phụ trách xử lý một nhóm dữ liệu đầu ra của Mapper. Việc xây dựng lớp Combiner tương tự như xây dựng lớp Reducer.

**Lớp Reducer**

Lớp Reducer hỗ trợ thực hiện quá trình Reduce. Tương tự như lớp Mapper, Lập trình viên sẽ thiết kế lớp mới thừa kế lại lớp Reducer, và định nghĩa lại các phương thức có sẵn nếu cần thiết, hai phương thức thường được định nghĩa lại là:

- Phương thức run(): Lập trình viên có thể định nghĩa lại phương thức này để kiểm soát việc đọc và phân phát dữ liệu từ quá trình Map gửi tới.

- Phương thức reduce(): Như phương thức map() trong lớp Mapper, Lập trình viên thường phải định nghĩa lại phương thức này, phương thức này được thiết kế để mỗi lần nhận vào và xử lý một loạt các cặp <key, value> có cùng chung thuộc tính key.

```
public static class Reduce extends  Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
                int sum = 0;
                while (values.hasNext()) {
                        sum += values.next().get();
                }
                output.collect(key, new IntWritable(sum));
        }
}
```

Ví dụ về sử dụng lớp Reducer trong chương trình Word count

**Lớp WritableComparator**

Dữ liệu được tạo ra từ tác vụ Map sau khi được phân nhóm, Combine và lưu trữ vào bộ nhớ cục bộ của các máy chạy map task, sẽ được các reduce task chép về bộ nhớ cục bộ của mình, mỗi reduce task chỉ chép về những dữ liệu thuộc nhóm được phân công xử lý. Tại đây, dữ liệu trước khi được xử lý tại phương thức reduce() sẽ được gom nhóm lại một lần nữa theo thuộc tính key và tổ chức sắp xếp trong từng nhóm nếu có yêu cầu. Lớp WritableComparator cho phép chúng ta định nghĩa lại hàm compare() tạo ra tiêu chí sắp xếp cho các cặp <key, value>. Nếu không khai báo và sử dụng lớp này thì mặc định các cặp <key, value> sau khi được gom nhóm sẽ không được sắp xếp theo bất kỳ tiêu chí nào.

```
public static class IntComparator extends WritableComparator {
        public IntComparator() {
                super(IntWritable.class);
        }

        @Override
        public int compare(byte[] b1, int s1, int l1,
        byte[] b2, int s2, int l2) {
                IntWritable key1=new IntWritable(0);
                IntWritable key2=new IntWritable(0);
                DataInputBuffer buffer = new DataInputBuffer();
                try
                {
                        buffer.reset(b1, s1, l1);
                        key1.readFields(buffer);
                        buffer.reset(b2, s2, l2);
                        key2.readFields(buffer);
                } catch (IOException e) {
                        throw new RuntimeException(e);
                }
                return key1.compareTo(key2)*-1;
        }
}
```

Ví dụ về sử dụng lớp WritableComparator để sắp xếp dữ liệu

### 2.3.2. Quy trình hoạt động

Khi Hadoop cluster được nạp một chương trình - một job, JobTracker sẽ thực hiện việc khởi tạo một job mới trên hệ thống. Nó sẽ đọc số lượng input file mà chương trình cần thực thi, thực hiện việc chia thành các input split. Tùy theo số lượng input split, JobTracker sẽ yêu cầu các TaskTracker khởi tạo đủ số lượng map task cần thiết cho việc xử lý.

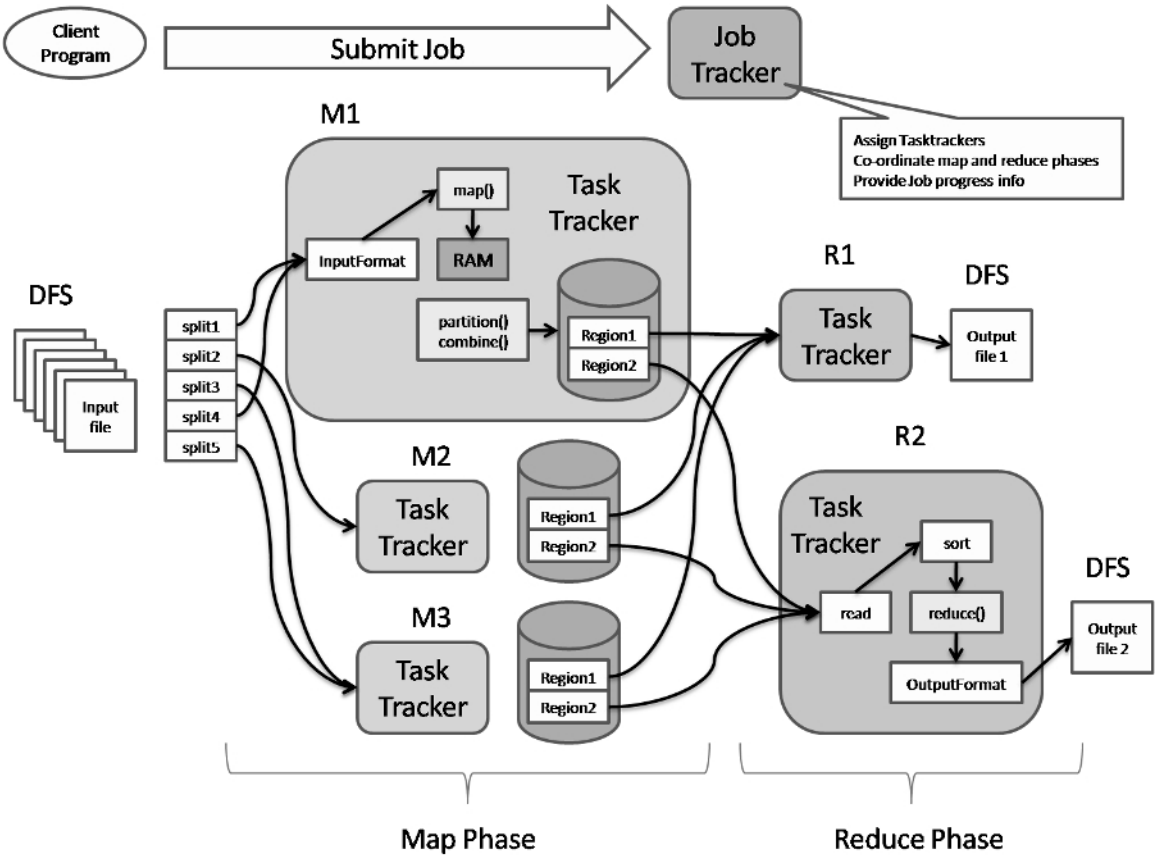
**Thực thi tại Map Task**

Mỗi map task sẽ đọc vào một input split và phân nó thành những record trong hàm run(), mỗi record là một cặp <key, value>. Sau đó, phương thức map() được gọi để thực hiện việc tính toán xử lý trên từng cặp <key, value>. Kết quả sau khi được xử lý sẽ không được chuyển ngay đến reduce task mà được lưu trữ tại bộ nhớ cục bộ của map task. Khi kích thước dữ liệu đạt đến ngưỡng quy định, map task thực hiện quá

tình Shuffle để phân nhóm dữ liệu. Nếu trong chương trình có thiết lập sử dụng lớp Combine, thì map task sẽ thực hiện việc Combiner cho từng nhóm dữ liệu. Kết quả sau khi thực hiện sẽ được ghi vào một tập tin tràn và đăng ký với TaskTracker. Khi kích thước tập tin đủ lớn sẽ thực hiện việc chuyển dữ liệu sang reduce task.

**Thực thi tại Reduce Task**

Đầu tiên reduce task sẽ chép dữ liệu từ các map task về bộ nhớ cục bộ của nó. Mỗi reduce task chỉ thực hiện việc chép những dữ liệu thuộc một nhóm nhất định. Tiếp theo, dữ liệu sẽ được gom nhóm theo key, mỗi nhóm có dạng <key, list(values)>, nếu được yêu cầu sắp xếp, dữ liệu trong mỗi nhóm sẽ được sắp xếp trước khi gửi qua phương thức reduce() để xử lý và ghi dữ liệu ra HDFS.




Hình 2.3.2.a: Quá trình hoạt động của một tác vụ MapReduce trên Hadoop

## 2.4. TỔNG QUAN HỆ THỐNG TÌM KIẾM

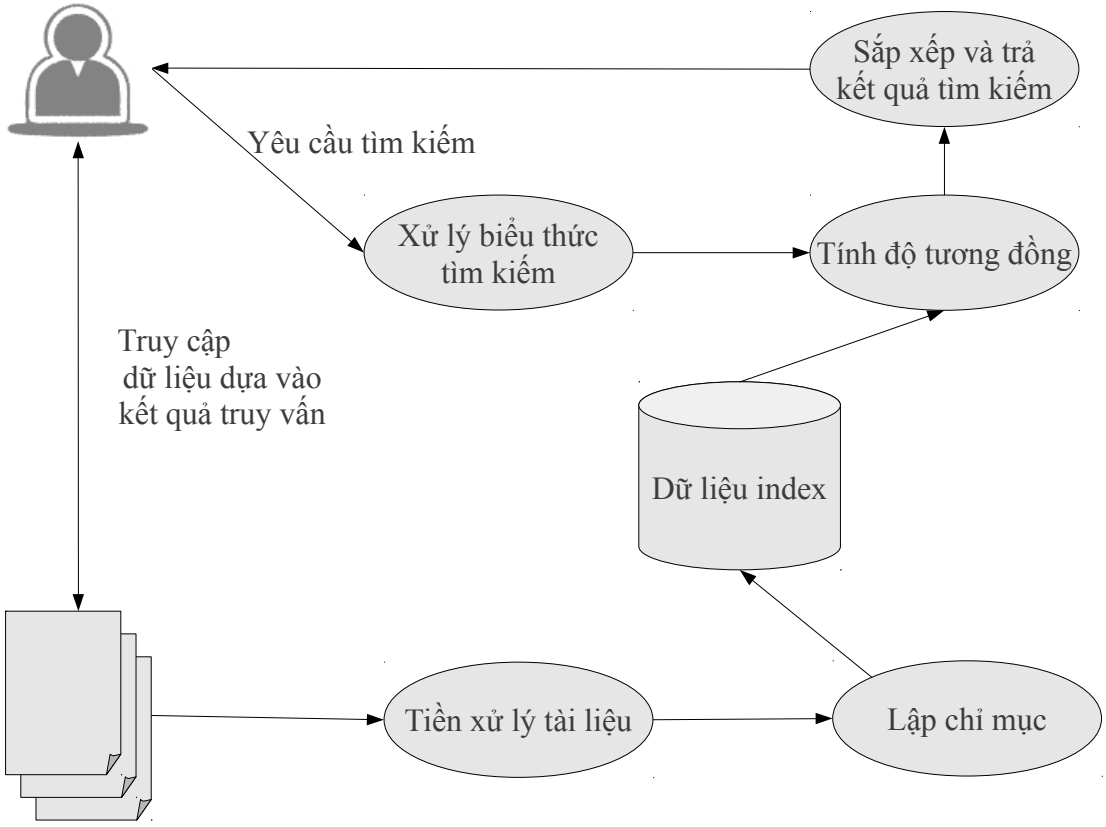
### 2.4.1. Giới thiệu chung

Với sự phát triển của máy tính, việc lưu trữ hàng trăm nghìn tài liệu không còn là vấn đề phức tạp. Tuy nhiên, người dùng thường không thể sử dụng hết kho dữ liệu đó. Họ chỉ quan tâm đến những tài liệu phù hợp với nhu cầu của mình. Việc xem qua tất cả các tài liệu là không khả thi dù có sự trợ giúp từ tốc độ xử lý của máy tính. Đồng thời, việc đánh giá tài liệu nào phù hợp với nhu cầu người dùng cũng là vấn đề rất phức tạp. Vì vậy, việc xây dựng hệ thống tìm kiếm dựa trên độ tương đồng giữa biểu thức tìm kiếm của người dùng với mỗi tài liệu trong tập tài liệu là rất cần thiết. Tuy nhiên, trên thực tế, rất khó để xác định mức độ liên quan giữa mỗi tài liệu với biểu thức tìm kiếm một cách trực tiếp, thay vào đó chúng ta có thể thực hiện gián tiếp bằng cách chuyển đổi các yếu tố thông tin của tài liệu và biểu thức tìm kiếm sang ngôn ngữ chỉ mục rồi sau đó mới xác định mức độ liên quan giữa chúng.

### 2.4.2. Các bước xây dựng hệ thống tìm kiếm

Công đoạn lập chỉ mục: Để có thể đưa vào lập chỉ mục, tập tài liệu phải trải qua quá trình tiền xử lý, tài liệu thô được xử lý thành các tài liệu được tách từ, phân đoạn và loại bỏ các yếu tố thông tin không quan trọng. Kết thúc quá trình tiền xử lý, các yếu tố thông tin trong tập tài liệu sẽ được tiến hành lập chỉ mục, tạo tiền đề cho việc tính độ tương đồng.

Công đoạn xử lý yêu cầu tìm kiếm: Người sử dụng có nhu cầu tìm kiếm đưa ra một biểu thức tìm kiếm phi cấu trúc bằng ngôn ngữ tự nhiên, mô tả nhu cầu thông tin của mình. Hệ thống tìm kiếm sẽ tiếp nhận và xử lý biểu thức tìm kiếm, biến đổi biểu thức tìm kiếm thành một tài liệu chỉ mục, tiếp theo hệ thống sẽ làm việc trên tập dữ liệu đã được lập chỉ mục trước đó kết hợp với tài liệu chỉ mục của biểu thức tìm kiếm, tính toán đưa ra một danh sách các tài liệu có liên quan đến biểu thức tìm kiếm, kèm theo chỉ số thể hiện mức độ liên quan, sắp xếp danh sách này theo chiều giảm dần của mức độ liên quan, ta được kết quả tìm kiếm.




Hình 2.4.2.a: Quy trình hoạt động của hệ thống tìm kiếm

### 2.4.3. Tần suất xuất hiện và trọng số của từ

**Tần suất xuất hiện của từ (Term Frequency)**

Truy vấn tự do trong quá trình tìm kiếm là dạng truy vấn có biểu thức tìm kiếm là một chuỗi ký tự, dùng mô tả nhu cầu tìm kiếm bằng ngôn ngữ tự nhiên, không có cấu trúc, không có bất kỳ toán tử kết nối nào (AND, OR, NOT, v.v). Dạng truy vấn này thường được sử dụng phổ biến trong lĩnh vực tìm kiếm web. Biểu thức tìm kiếm của truy vấn tự do thường là tập hợp một số từ khóa, có liên quan đến vấn đề cần tìm kiếm.

Để đánh giá mức độ tương đồng giữa các tài liệu và biểu thức tìm kiếm, người ta thường dựa vào số lần xuất hiện của các từ khóa trong biểu thức tìm kiếm trên mỗi tài liệu. Một từ xuất hiện càng nhiều lần trong tài liệu thì tài liệu càng có liên quan mật thiết với từ đó. Như vậy, để đánh giá mức độ liên quan của từ T trong biểu thức tìm kiếm Q với tài liệu D, ta sẽ dựa vào số lần xuất hiện của T trong D.

Tuy nhiên, tất cả các từ trong mỗi tài liệu không phải lúc nào cũng có độ quan trọng và cần được quan tâm như nhau. Các từ nằm trong danh sách stop word là những từ xuất hiện trong hầu hết tất cả các tài liệu và xuất hiện rất nhiều lần trong mỗi tài liệu, nên nó sẽ không được dùng để đánh giá mức độ liên quan và đại diện cho nội dung của tài liệu. Các stop word sẽ bị loại bỏ trong quá trình xử lý. Ngoài ra, còn một vấn đề khác liên quan đến độ dài của tài liệu, ví dụ: một từ xuất hiện 30 lần trong một tài liệu 1000 từ so với một từ xuất hiện 30 lần trong một tài liệu 100000 từ rõ ràng mức độ quan trọng của chúng là khác nhau. Vì vậy, để khắc phục vấn đề này, công thức chuẩn hóa việc tính tần suất xuất hiện của từ i trong tài liệu j được thành lập như sau:

$$tf_{ij} = \frac{t_{ij}}{d_j}$$

Trong công thức trên:

- $tf_{ij}$ là tần suất xuất hiện của từ i trong tài liệu j.
- $t_{ij}$ là số lần xuất hiện của từ i trong tài liệu j.
- $d_j$ là tổng số từ trong tài liệu j.

**Tần suất nghịch đảo tài liệu (Inverst Document Frequency)**

Đại lượng IDF (Inverst Document Frequency) thể hiện độ hiếm của từ T trên toàn bộ tập tài liệu. Từ T xuất hiện trong càng ít tài liệu, điều đó có nghĩa là khi nó xuất hiện trong tài liệu D thì nó là điểm quan trọng để phân biệt tài liệu D với các tài liệu khác và hàm lượng thông tin trong nó càng lớn. Công thức tính IDF như sau:

$$idf_i = \log\left(\frac{N}{n_i}\right)$$

Trong công thức trên:

- $idf_i$ là tần suất nghịch đảo tài liệu của từ i.
- $N$ là tổng số tài liệu có trong tập tài liệu.
- $n_i$ là tổng số tài liệu có chứa từ i.

**Trọng số tf-idf**

Trọng số tf-idf là sự tổng hợp của chỉ số tần suất xuất hiện và tần suất nghịch đảo tài liệu của từ. Trọng số tf-idf thể hiện mức độ quan trọng của mỗi từ với mỗi tài liệu, được xét trên ngữ cảnh của tập tài liệu. Trọng số tf-idf của từ i trong tài liệu j được tính như sau:

$$tf-idf_{ij} = tf_{ij} * idf_i$$

Trong công thức trên:

- $tf-idf_{ij}$ là trọng số tf-idf của từ i trong tài liệu j.
- $tf_{ij}$ là tần suất xuất hiện của từ i trong tài liệu j.
- $idf_i$ là tần suất đảo ngược tài liệu của từ i trong tập tài liệu.

### 2.4.4. Mô hình không gian vectơ dùng tính độ tương đồng

**Giới thiệu chung**

Để đánh giá mức độ liên quan giữa các tài liệu, người ta thường dùng mô hình không gian vector, đây cũng là mô hình thông dụng nhất trong tìm kiếm thông tin.

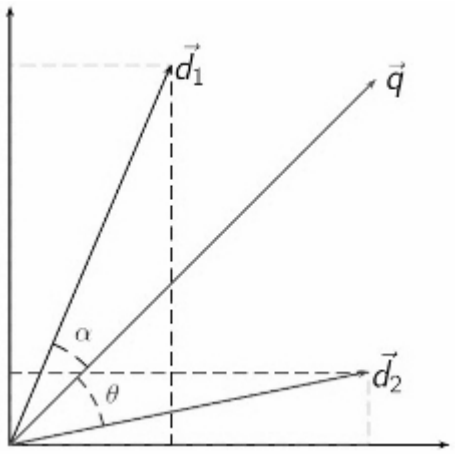
Trong mô hình này, mỗi tài liệu được biểu diễn bằng một vector trong một không gian có số chiều lớn, trong đó mỗi chiều của không gian tương ứng với một từ trong tập tài liệu.

Với tập tài liệu chứa n từ khác nhau, không gian vector dùng biểu diễn các tài liệu sẽ có n chiều, tọa độ của mỗi vector tài liệu sẽ là một tập hợp n phần tử. Giả sử tài liệu d được biểu diễn bằng vector $\vec{V}(d)$ , thì mỗi phần tử xác định tọa độ của vector $\vec{V}(d)$ là chỉ số tf-idf của một từ trong tài liệu d tương ứng với một chiều trong không gian vector, nếu từ không xuất hiện trong tài liệu d, ta gán trọng số tf-idf bằng 0 cho từ đó. Cách trình bày này không quan tâm đến thứ tự của các từ trong mỗi tài liệu. Ví dụ, hai tài liệu lần lược có nội dung là "xã hội phát triển" và "phát triển xã hội" là như nhau, khi biểu diễn thành vector.

Để xác định độ tương đồng giữa hai vector tài liệu trên mô hình không gian vector, công thức Cosine được đề nghị sử dụng. Giả sử, ta cần tính độ tương đồng giữa hai vector tài liệu $\vec{V}(d_1)$ và $\vec{V}(d_2)$ , áp dụng công thức Cosine ta có:

$$Similarity(d_1, d_2) = \frac{\vec{V}(d_1).\vec{V}(d_2)}{|\vec{V}(d_1)|.|\vec{V}(d_2)|}$$

Trong công thức trên: tử số là tích vô hướng của hai vector $\vec{V}(d_1)$ và $\vec{V}(d_2)$ . Mẫu số là tích độ dài của hai vector $\vec{V}(d_1)$ và $\vec{V}(d_2)$ .




Hình 2.4.4.a: Minh họa độ tương đồng - Cosine giữa các vector

### 2.4.5. Ví dụ minh họa

Giả sử có một tập tài liệu chứa ba tài liệu với nội dung của mỗi tài liệu như sau:

- d1: ”new york times”
- d2: ”new york post”
- d3: ”los angeles times”

**Công đoạn lập chỉ mục:**

Áp dụng công thức, ta tính được chỉ số idf cho mỗi từ trong tập tài liệu:

$idf_{angles} = \log(3/1) = 0.477$

$idf_{los} = \log(3/1) = 0.477$

$idf_{new} = \log(3/2) = 0.176$

$idf_{post} = \log(3/1) = 0.477$

$idf_{times} = \log(3/2) = 0.176$

$idf_{york} = \log(3/2) = 0.176$

Chỉ số tf của mỗi từ trong mỗi tài liệu:

| | angeles | los | new | post | times | york |
|---|---|---|---|---|---|---|
| d1 | null | null | 1/3 | null | 1/3 | 1/3 |
| d2 | null | null | 1/3 | 1/3 | null | 1/3 |
| d3 | 1/3 | 1/3 | null | null | 1/3 | Null |

Kết hợp các chỉ số idf, tf đã tính ở trên, ta có bảng trọng số tf-idf như sau:

| | angeles | los | new | post | times | york |
|---|---|---|---|---|---|---|
| d1 | null | null | 0.058 | null | 0.058 | 0.058 |
| d2 | null | null | 0.058 | 0.159 | null | 0.058 |
| d3 | 0.159 | 0.159 | null | null | 0.058 | null |

**Công đoạn xử lý yêu cầu tìm kiếm**

Trong tập tài liệu trên có chứa tổng số 6 từ khác nhau. Như vậy, không gian vector dùng biểu diễn các tài liệu trong tập tài liệu này sẽ có 6 chiều.

Ta tính được độ dài của các vectơ tài liệu như sau:

$$|\vec{V}(d_1)| = \sqrt{0^2+0^2+0.058^2+0^2+0.058^2+0.058^2} = 0.100$$

$$|\vec{V}(d_2)| = \sqrt{0^2+0^2+0.058^2+0.159^2+0^2+0.058^2} = 0.178$$

$$|\vec{V}(d_3)| = \sqrt{0.159^2+0.159^2+0^2+0^2+0.058^2+0^2} = 0.232$$

Xét yêu cầu tìm kiếm với biểu thức tìm kiếm q có nội dung ”new new times”. Để xác định được độ tương đồng của q với các tài liệu trong tập tài liệu, đầu tiên, ta cần tính chỉ số tf-idf cho các từ trong q. Xem q như một tài liệu thông thường, kết hợp với chỉ số idf được lấy từ kho dữ liệu idf tính trước đó. Ta tính được tf-idf như sau:

| | angeles | los | new | post | times | york |
|---|---|---|---|---|---|---|
| q | null | null | (0.176 *2/3) = 0.117 | null | (0.176 *1/3) = 0.058 | null |

Độ dài vectơ biểu thức tìm kiếm là:

$$|\vec{V}(q)| = \sqrt{0.117^2+0.058^2} = 0.130$$

Độ tương đồng giữa biểu thức tìm kiếm với các tài liệu trong tập tài liệu được tính như sau:

$$Similarity(q, d_1) = \frac{0.117x0.058 + 0.058x0.058}{0.100x0.130} = 0.780$$

$$Similarity(q, d_2) = \frac{0.117x0.058 + 0.058x0}{0.178x0.130} = 0.293$$

$$Similarity(q, d_3) = \frac{0.117x0 + 0.058x0.058}{0.232x0.130} = 0.111$$

Vậy thứ tự độ tương đồng của các tài liệu trong tập tài liệu với biểu thức tìm kiếm q là d1 > d2 > d3. Kết quả tìm kiếm là danh sách có thứ tự sau: {d1, d2, d3}.

# CHƯƠNG 3: ỨNG DỤNG CƠ SỞ LÝ THUYẾT, GIẢI QUYẾT VẤN ĐỀ

Với mục tiêu và phạm vi nghiên cứu đã được nêu ở phần trên, trong phần này chỉ tập trung trình bày nội dung giải quyết vấn đề xử lý yêu cầu tìm kiếm.

## 3.1. NGỮ CẢNH GIẢI QUYẾT VẤN ĐỀ

Việc giải quyết vấn đề xử lý yêu cầu tìm kiếm được đặt trong ngữ cảnh: hệ thống lập chỉ mục đã được xây dựng xong, tập tài liệu dùng cho tìm kiếm đã được lập chỉ mục tạo ra tập tài liệu chỉ mục hoàn chỉnh, trong đó:

- Thông tin về chỉ số idf của tập tài liệu được lưu thành nhiều record, mỗi record lưu trữ thông tin chỉ số idf của một từ có xuất hiện trong tập tài liệu. Như vậy, mỗi record sẽ gồm có hai trường là: từ và chỉ số idf tương ứng với từ đó. Về mặt tổ chức lưu trữ trên HDFS, mỗi record sẽ được tổ chức lưu trữ dưới dạng một tập tin plain-text, với tên của tập tin là giá trị hashCode của từ, nội dung của tập tin là một số thực biểu diễn giá trị idf tương ứng với từ.

- Thông tin về trọng số tf-idf của tập tài liệu được lưu thành nhiều record, mỗi record chứa thông tin tf-idf của một từ trong một tài liệu cụ thể nào đó thuộc tập tài liệu. Như vậy, mỗi record sẽ gồm có ba trường: tên tài liệu, từ, chỉ số tf-idf của từ ở trường thứ hai trong tài liệu có tên được nêu ở trường thứ nhất. Về mặt tổ chức lưu trữ trên HDFS, toàn bộ dữ liệu về trọng số tf-idf của tập tài liệu sẽ được lưu trong một tập tin plain-text duy nhất, mỗi record sẽ nằm trên một dòng trong tập tin plain-text này.

- Thông tin về độ dài của các vector tài liệu được lưu thành nhiều record, mỗi record bao gồm tên tài liệu, kèm theo chỉ số độ dài vector tương ứng với tài liệu đó. Toàn bộ dữ liệu về độ dài vector của các tài liệu trong tập tài liệu sẽ được lưu trong một tập tin plain-text duy nhất, mỗi record sẽ nằm trên một dòng trong tập tin plain-text này.

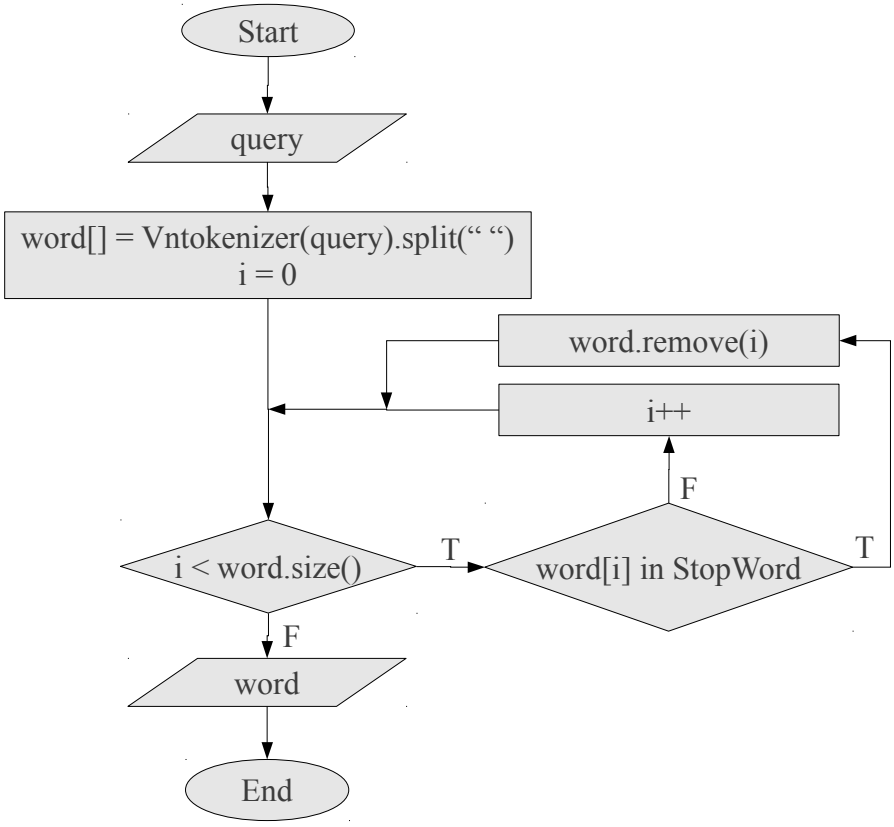
## 3.2. GIẢI THUẬT

Quá trình xử lý yêu cầu tìm kiếm có thể được chia ra làm ba quá trình con: xử

lý biểu thức tìm kiếm, tính độ tương đồng, sắp xếp các tài liệu theo độ tương đồng và xuất kết quả tìm kiếm. Ba quá trình này diễn ra một cách tuần tự theo đúng thứ tự đã nêu. Sau đây là các giải thuật được sử dụng trong mỗi quá trình.

### 3.2.1. Xử lý biểu thức tìm kiếm

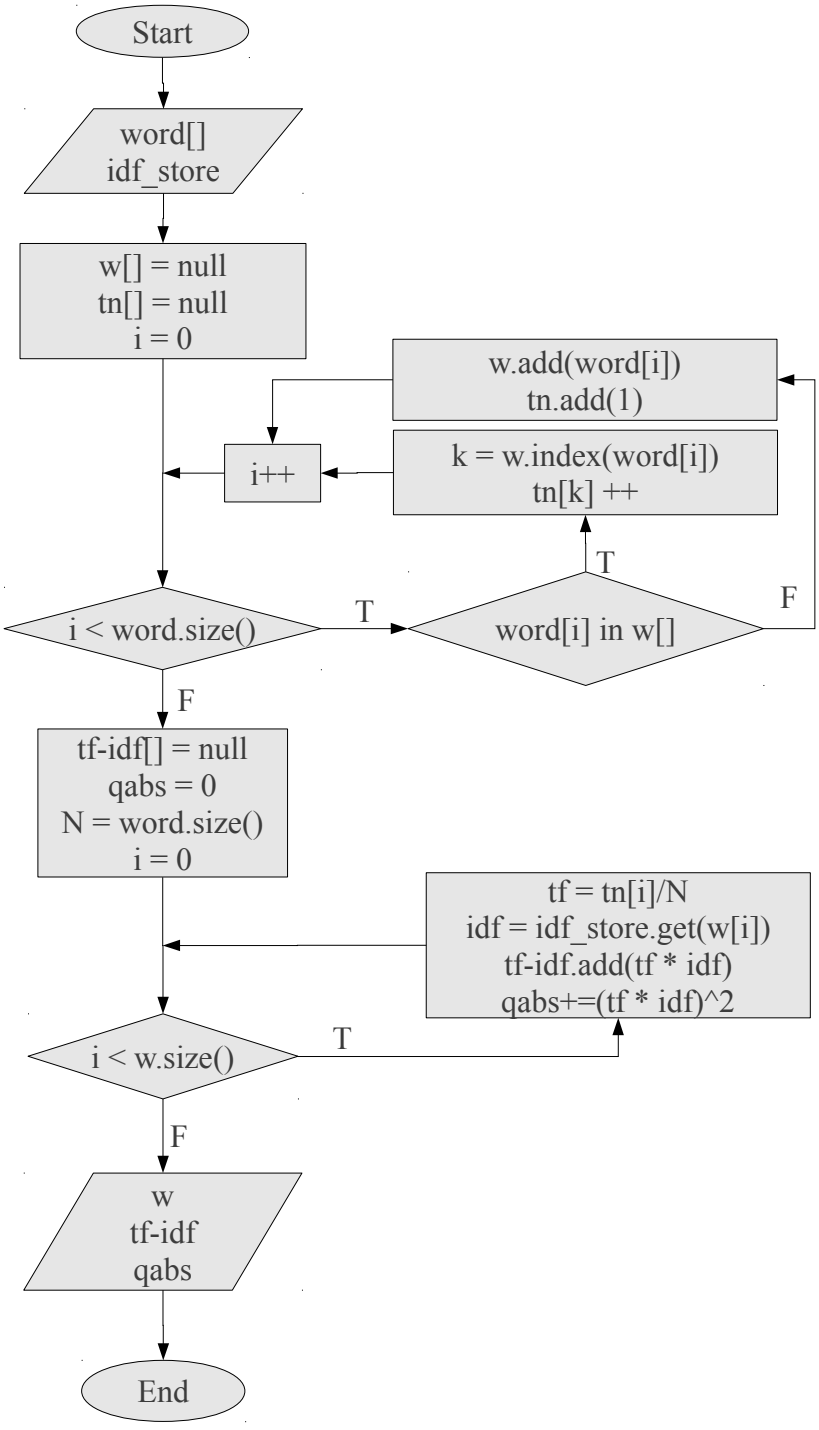
Có thể coi biểu thức tìm kiếm là một tài liệu văn bản bất cấu trúc, do người có nhu cầu tìm kiếm cung cấp, để có thể tính độ tương đồng giữa biểu thức tìm kiếm với các vector tài liệu, biểu thức tìm kiếm cần được xử lý tách từ, loại stop word, lập chỉ mục như một tài liệu thông thường. Sau các bước này ta mới có thể biểu diễn biểu thức tìm kiếm thành một vector để áp dụng công thức Cosine tính độ tương đồng. Sau đây là các giải thuật được sử dụng trong giai đoạn xử lý biểu thức tìm kiếm:




Hình 3.2.1.a: Lưu đồ giải thuật tách từ, loại stop word cho biểu thức tìm kiếm

Trong lưu đồ trên: Hàm VNTokenizer() được dùng trong lưu đồ là để giải quyết vấn đề tách từ cho biểu thức tìm kiếm. Xét về mặt hình thức, trong một đoạn văn bản tiếng Việt, các từ đơn (từ gồm một tiếng duy nhất) được phân cách với nhau bằng ký

tự space, dựa vào đặc điểm này, các từ đơn có thể dễ dàng được phân tách. Tuy nhiên, với từ ghép (từ gồm nhiều tiếng) thì phương pháp trên hoàn toàn không khả thi. Vấn đề tách từ trong văn bản tiếng Việt đã được đặt ra từ lâu, đã có nhiều nghiên cứu tìm cách giải quyết vấn đề này. Một trong số đó là đề tài nghiên cứu công cụ VNTokenizer của tác giả Lê Hồng Phương [2]. Mỗi từ trong văn bản sau khi được xử lý bởi VNTokenize sẽ được tách biệt với nhau bằng ký tự space, còn các tiếng trong một từ (trường hợp từ có nhiều tiếng) sẽ được tách biệt với nhau bằng ký tự "_". Trong luận văn này, quá trình xử lý tách từ, phân đoạn được thực hiện nhờ gói VNTokenizer đã nêu trên.

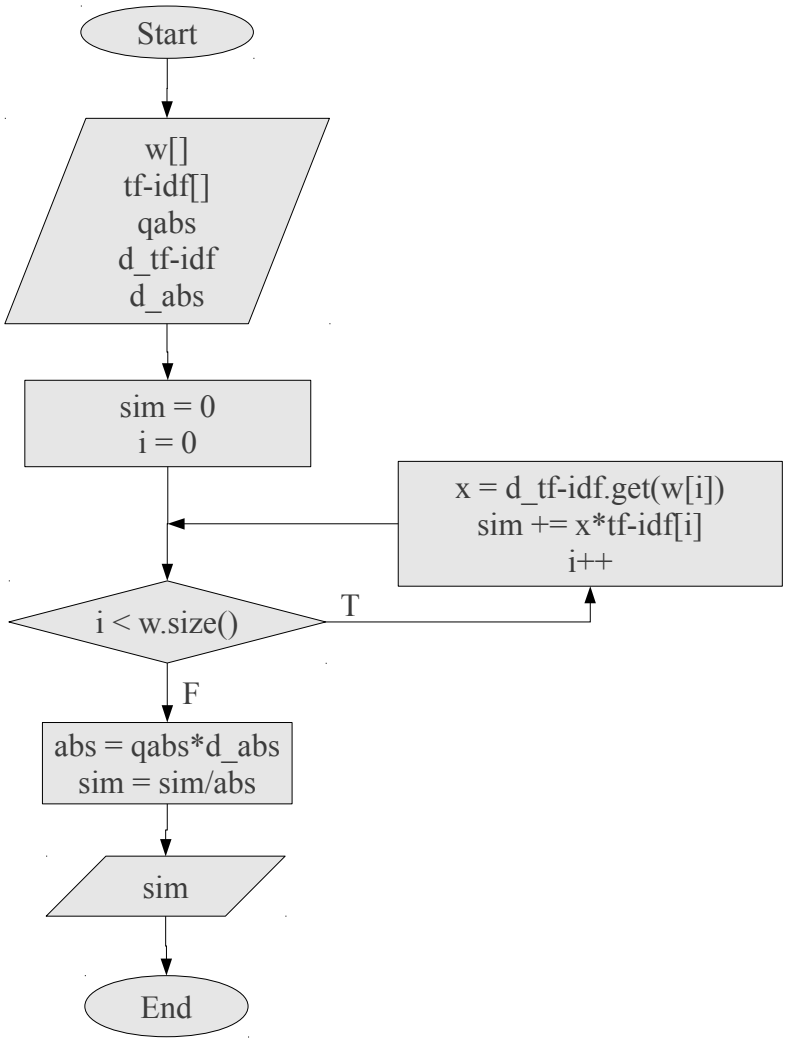



Hình 3.2.1.b: Lưu đồ giải thuật lập chỉ mục cho biểu thức tìm kiếm

### 3.2.2. Tính độ tương đồng

Sau khi biểu thức tìm kiếm đã được xử lý, ta áp dụng công thức Cosine, lần lượt tính độ tương đồng của biểu thức tìm kiếm với mỗi tài liệu trong tập tài liệu. Sau

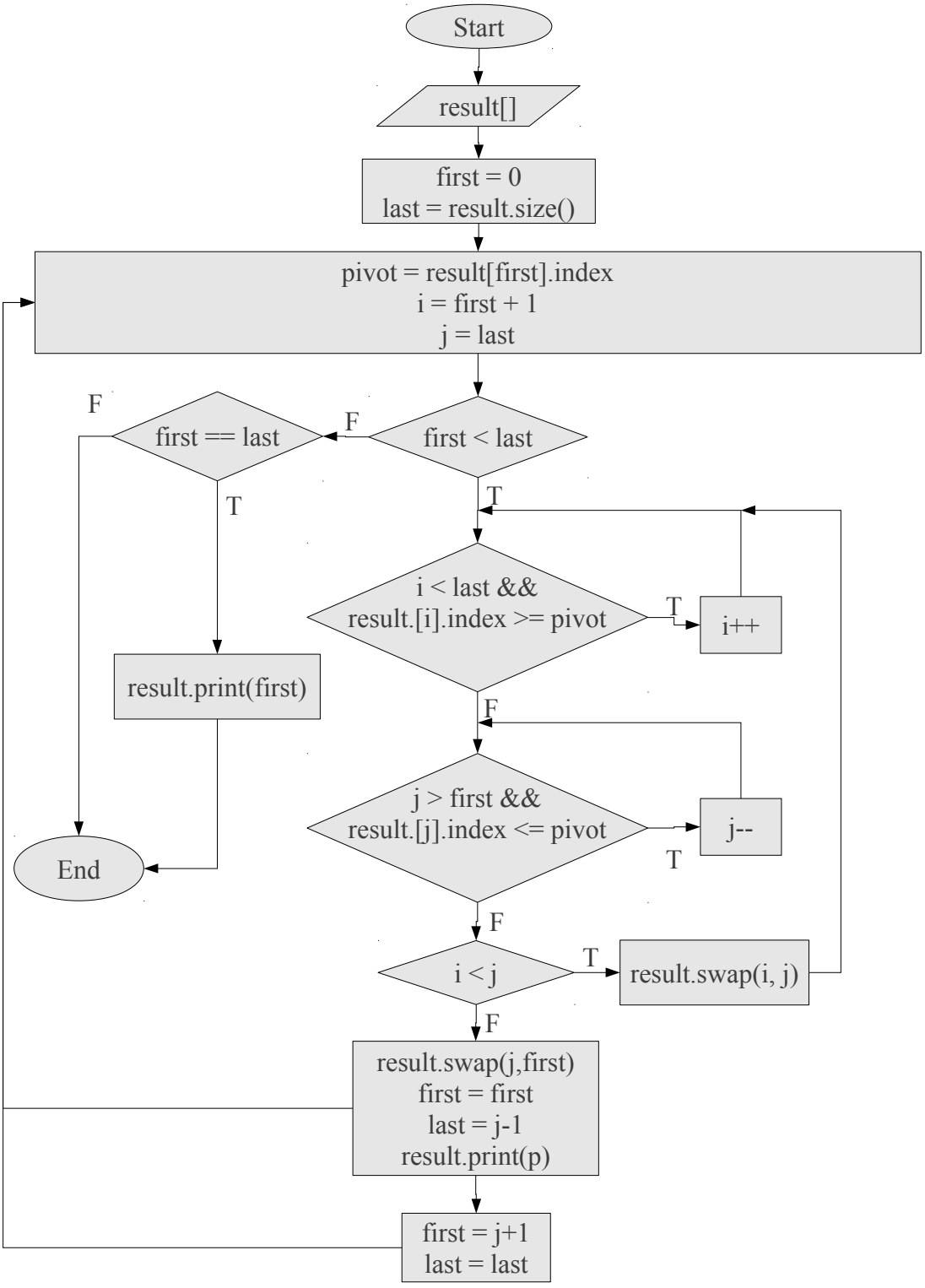
đây là giải thuật tính độ tương đồng của biểu thức tìm kiếm với một tài liệu:




Hình 3.2.2.a: Lưu đồ giải thuật tính độ tương đồng giữa biểu thức tìm kiếm với tài liệu

### 3.2.3. Sắp xếp tài liệu theo độ tương đồng và xuất kết quả tìm kiếm

Kết thúc quá trình tính độ tương đồng, ta có được danh sách các tài liệu có liên quan đến biểu thức tìm kiếm, kèm theo độ tương đồng tương ứng với mỗi tài liệu. Sắp xếp danh sách này theo thứ tự giảm dần của độ tương đồng, ta có được kết quả tìm kiếm.




Hình 3.2.3.a: Lưu đồ giải thuật sắp xếp độ tương đồng và xuất kết quả tìm kiếm

Trong lưu đồ trên: Giải thuật sắp xếp được sử dụng là giải thuật Quick sort. Xét một cách tổng quát, khi áp dụng giải thuật này dữ liệu sẽ được sắp xếp dần theo chiều

từ đầu đến cuối danh sách, tức là những dữ liệu càng về đầu danh sách sẽ được ưu tiên sắp xếp trước. Trong lưu đồ trên, những dữ liệu đã xác định được vị trí, sẽ được xuất ra ngay lập tức, chứ không cần đợi toàn bộ quá trình sắp xếp thực hiện xong.

## 3.3. ỨNG DỤNG HADOOP - TĂNG TỐC ĐỘ XỬ LÝ CỦA HỆ THỐNG

Đánh giá thời gian thực hiện và xét khả năng ứng dụng Hadoop vào ba quá trình con của quá trình xử lý yêu cầu tìm kiếm.

### 3.3.1. Xử lý biểu thức tìm kiếm

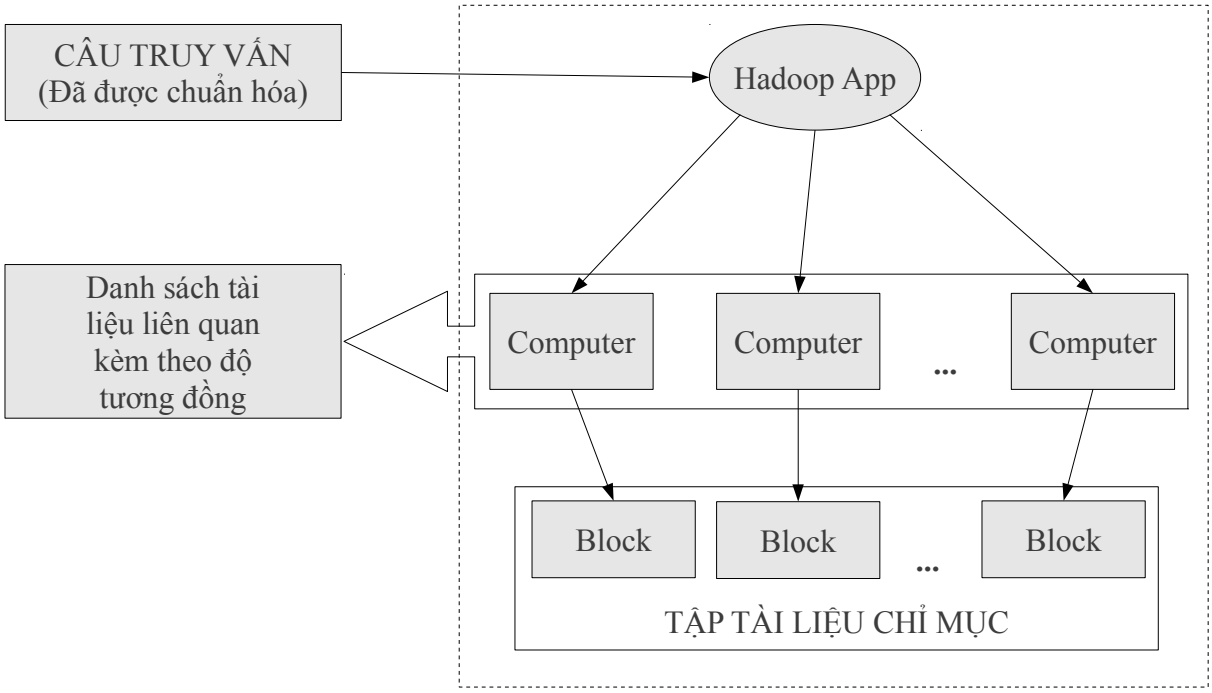
Xét trong nội bộ quá trình xử lý một yêu cầu tìm kiếm: giai đoạn này tương đối đơn giản, số lượng phép toán cần phải thực hiện không nhiều, có thể áp dụng giải thuật tuần tự để thực hiện.

Xét tổng quát cả hệ thống: Trường hợp hệ thống tiếp nhận một lúc nhiều yêu cầu tìm kiếm, nếu thực hiện việc xử lý lần lượt từng biểu thức tìm kiếm một, thì những yêu cầu tìm kiếm đến sau sẽ phải chờ, nếu số lượng yêu cầu tìm kiếm đến nhiều, thời gian chờ sẽ rất lâu. Với một hệ thống có số lượng người dùng lớn, cần áp dụng kỹ thuật xử lý phân tán, song song cho giai đoạn này, khi đó các biểu thức tìm kiếm sẽ được phân phát ra cho nhiều máy tính cùng xử lý. Tuy nhiên, xét quy mô của hệ thống tìm kiếm này, chúng ta chỉ cần áp dụng kỹ thuật đa luồng để thực hiện xử lý các biểu thức tìm kiếm đến, mỗi biểu thức tìm kiếm do một luồng phụ trách.

### 3.3.2. Tính độ tương đồng

Xét công việc tính độ tương đồng cho một tài liệu: Với giải thuật tính độ tương đồng cho mỗi tài liệu đã được nêu ở trên, ta xét thấy số lượng phép toán của công việc này không lớn, nên không cần phải áp dụng kỹ thuật xử lý phân tán, song song để thực hiện.

Xét trên cả hệ thống: Với quy mô của tập tài liệu chỉ mục, việc tính độ tương đồng của toàn bộ tài liệu với biểu thức tìm kiếm cần được áp dụng kỹ thuật xử lý phân tán, song song. Tập tài liệu chỉ mục sẽ được chia ra làm nhiều phần, mỗi phần gồm thông tin chỉ mục của một số tài liệu, mỗi máy tính sẽ được giao nhiệm vụ tính độ tương đồng cho các tài liệu trong một phần.




Hình 3.3.2.a: Mô hình tính độ tương đồng

Vấn đề phân chia tập tài liệu chỉ mục: Giai đoạn tính độ tương đồng chỉ sử dụng dữ liệu trọng số tf-idf và độ dài vector tài liệu của tập tài liệu chỉ mục. Nên ta chỉ thực hiện phân chia cho các dữ liệu này. Việc phân chia tập tài liệu chỉ mục phải đảm bảo tính toàn vẹn dữ liệu - tất cả dữ liệu chỉ mục của mỗi tài liệu phải đảm bảo chỉ nằm trên một block duy nhất. Giải pháp đưa ra là dựa vào chỉ số hashCode của tên tài liệu, cụ thể như sau: Giả sử ta muốn chia tập tài liệu chỉ mục ra làm n phần. Ở mỗi record của dữ liệu tf-idf và độ dài vector tài liệu, ta lấy ra thông số tên tài liệu, tiếp theo ta hashCode tên tài liệu và chia cho n, lấy phần dư (p = hashCode(document name) % n), số dư đó chính là số hiệu của block mà record đang xét sẽ được chuyển đến lưu trữ.

### 3.3.3. Sắp xếp tài liệu theo độ tương đồng và xuất kết quả tìm kiếm

Số lượng tài liệu có liên quan đến biểu thức tìm kiếm là một giá trị có độ dao động rất lớn tùy theo nội dung của biểu thức tìm kiếm, và kích cở của tập tài liệu chỉ mục. Đối với các hệ thống lớn, để đảm bảo tốc độ, cần tổ chức xử lý phân tán, song song công việc này, mỗi máy tính sẽ chịu trách nhiệm sắp xếp các tài liệu có độ tương đồng nằm trong một khoảng nhất định. Với quy mô của hệ thống tìm kiếm này, việc sắp xếp danh sách các tài liệu liên quan của mỗi yêu cầu tìm kiếm sẽ được thực hiện

tuần tự, nếu có nhiều yêu cầu sắp xếp xuất hiện cùng một lúc, hệ thống sẽ chia ra cho nhiều máy, mỗi máy sắp xếp một danh sách.

## 3.4. CÁC ĐỀ XUẤT VÀ CẢI TIẾN.

### 3.4.1. Cải tiến công thức tính độ tương đồng

Trong một hệ thống tìm kiếm tài liệu có xếp hạng, việc tính độ tương đồng giữa các tài liệu với biểu thức tìm kiếm là nhằm tạo ra cơ sở để xếp hạng các tài liệu. Như vậy, chúng ta có thể lược bỏ một số thành phần trong công thức Cosine - dùng tính độ tương đồng, để giảm số lượng phép tính phải thực hiện, miễn sao không làm thay đổi kết quả xếp hạng các tài liệu.

Giả sử, sau khi tính được độ tương đồng giữa các tài liệu d1, d2 với biểu thức tìm kiếm q ta có bất đẳng thức sau:

$$Similarity(d_1, q) > Similarity(d_2, q)$$

$$<=> \quad \frac{\vec{V}(d_1).\vec{V}(q)}{|\vec{V}(d_1)|.|\vec{V}(q)|} > \frac{\vec{V}(d_2).\vec{V}(q)}{|\vec{V}(d_2)|.|\vec{V}(q)|}$$

$$<=> \quad \frac{\vec{V}(d_1).\vec{V}(q)}{|\vec{V}(d_1)|} > \frac{\vec{V}(d_2).\vec{V}(q)}{|\vec{V}(d_2)|}$$

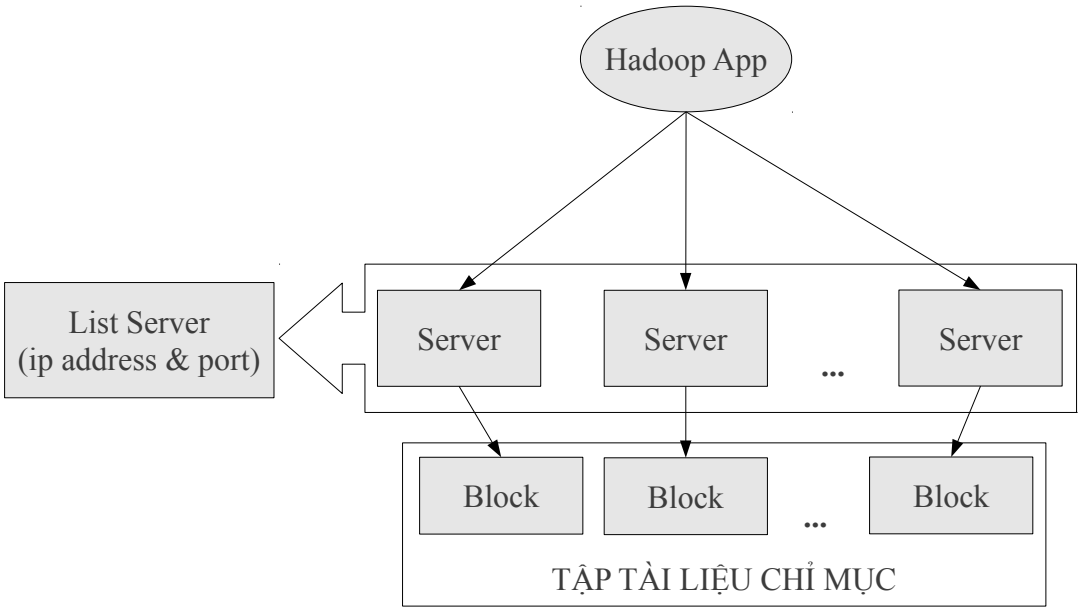
Rõ ràng việc nhân vào hai vế của bất đẳng thức trên một giá trị $|\vec{V}(q)|$ sẽ không làm đổi dấu của bất đẳng thức. Đều này cũng có nghĩa là khi áp dụng công thức Cosine vào việc tính độ tường đồng giữa các tài liệu với biểu thức tìm kiếm, ta có thể lược bỏ giá trị độ dài vector biểu thức tìm kiếm ra khỏi mẫu số của công thức, mà không sợ làm thay đổi kết quả xếp hạng các tài liệu. Khi đó công thức tính độ tương đồng giữa tài liệu d và biểu thức tìm kiếm q sẽ là:

$$Similarity(d, q) = \frac{\vec{V}(d).\vec{V}(q)}{|\vec{V}(d)|}$$

### 3.4.2. Khắc phục khuyết điểm của Hadoop trong tính độ tương đồng

Nếu sử dụng mô hình nguyên mẫu của Hadoop, việc tính độ tương đồng giữa biểu thức tìm kiếm với các tài liệu trong tập tài liệu là một job. Để tính toán, job cần tiến hành đọc các dữ liệu cần thiết từ block, kết thúc một job, toàn bộ bộ nhớ đệm dành cho job đó sẽ được giải phóng, khi một yêu cầu tính độ tương đồng khác xuất hiện, job mới được khởi tạo phải một lần nữa xuống ổ cứng để đọc block, dữ liệu đọc lên sẽ có nhiều phần giống với dữ liệu mà các job trước đó đã đọc lên, việc đọc dữ liệu như vậy tạo nên một sự lãng phí tài nguyên, chưa kể việc đọc dữ liệu trên ổ cứng mất rất nhiều thời gian. Vậy, vấn đề thứ nhất được đặt ra là ứng dụng Hadoop nguyên mẫu mất quá nhiều thời gian để đọc dữ liệu. Vấn đề thứ hai được đặt ra là Hadoop mất một khoảng thời gian khá lâu (khoảng 10 giây) để khởi tạo một job.

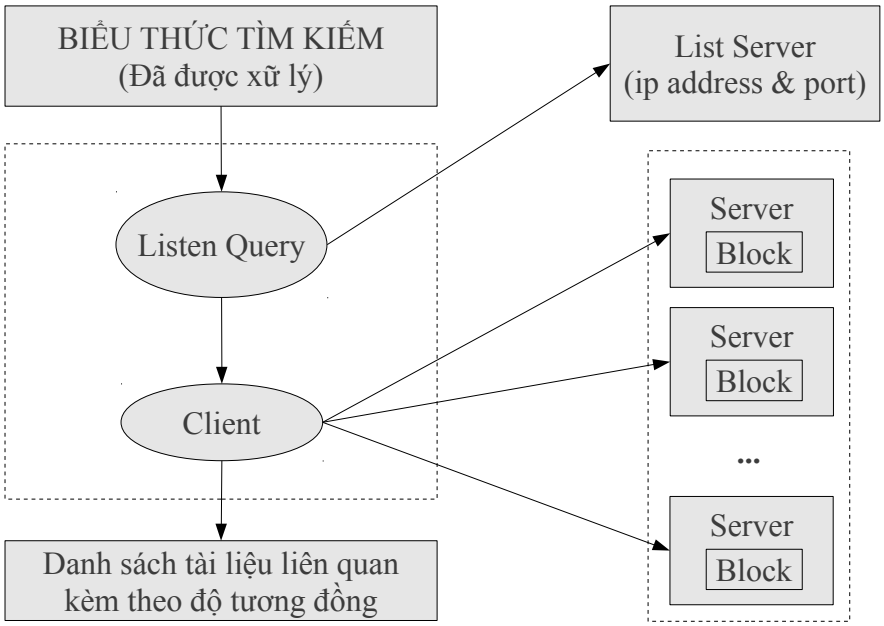
Cả hải vấn đề đã nêu ở trên sẽ được giải quyết nếu sử dụng một job duy nhất cho tất cả các yêu cầu tính độ tương đồng. Khi đó, chúng ta phải tìm một cách nào đó để tách biệt kết quả tính độ tương đồng của mỗi yêu cầu tìm kiếm. Mô hình server/client được đề xuất sử dụng.




Hình 3.4.2.a: Mô hình ứng dụng Hadoop, khởi tạo các server dùng tính độ tương đồng

Trong mô hình trên: Các server được khởi tạo sẽ đọc toàn bộ block do mình phụ trách lên bộ nhớ đệm và duy trì bộ nhớ đó cho đến khi nào server được tắt. Khi có

yêu cầu tính độ tương đồng được gửi tới, server sẽ sử dụng dữ liệu trên bộ nhớ đệm để tính toán, việc truy cập dữ liệu trên bộ nhớ đệm nhanh hơn nhiều lần so với việc truy cập dữ liệu trên ổ cứng. Như vậy, vấn đề về thời gian đọc dữ liệu của mô hình Hadoop nguyên mẫu đã được giải quyết. Đồng thời, việc duy trì các server cũng giải quyết được vấn đề về chi phí thời gian khởi tạo job. Vấn đề còn lại cần phải giải quyết là làm thế nào để phân biệt kết quả tính độ tương đồng của mỗi yêu cầu tìm kiếm.




Hình 3.4.2.b: Mô hình server/client dùng tính độ tương đồng

Trong mô hình trên: Quá trình tính độ tương đồng cho mỗi yêu cầu tìm kiếm sẽ được bắt đầu từ việc khởi tạo một client, client này sẽ thiết lập kết nối đến tất cả các server, quá trình trao đổi dữ liệu giữa client và các server sẽ được thực hiện trên kết nối đã tạo. Sau khi hoàn tất việc tạo kết nối, client sẽ gửi yêu cầu tính độ tương đồng đến tất cả các server, và chờ kết quả phản hồi, ngay khi nhận được đầy đủ kết quả phản hồi, client sẽ đóng các kết nối, tổng hợp kết quả chuyển sang cho giai đoạn sắp xếp và xuất kết quả tìm kiếm, sau đó tự giải phóng. Về phía server, mỗi server sau khi chấp nhận kết nối từ client sẽ nhận và thực hiện yêu cầu tính độ tương đồng từ client gửi tới, sau đó trả kết quả về theo nguyên tắc: nhận yêu cầu trên kết nối nào thì trả kết quả về trên kết nối đó. Như vậy, cho dù trong cùng một lúc, server đồng thời được kết nối từ nhiều client, thì server vẫn có cơ sở để trả kết quả về đúng nơi. Vấn đề về phân biệt kết quả tính độ tương đồng của mỗi yêu cầu tìm kiếm đã được giải quyết.

Với mô hình server/client - dùng tính độ tương đồng đã được trình bài ở trên, ta hoàn toàn có thể sử dụng kỹ thuật lập trình socket để cài đặt. Các socket server sau khi được khởi tạo sẽ gửi thông tin socket của mình lên một danh sách trên hệ thống HDFS, mỗi socket client sẽ dựa vào danh sách này để khởi tạo các kết nối đến socket server, quá trình trao đổi dữ liệu giữa socket client và socket server sẽ được thực hiện thông qua kết nối đã tạo.

### 3.4.3. Cải tiến cách lưu trữ trọng số tfi-df trên mỗi block trong quá trình tính độ tương đồng

Giả sử ta cần tính độ tương đồng giữa biểu thức tìm kiếm q - "new york" và tài liệu d - "new york times".

Xét công thức tính độ tương đồng, thì rõ ràng trọng số tf-idf của từ "times" trong tài liệu d không có ý nghĩa trong kết quả của phép tính độ tương đồng này. Vấn đề đặt ra là làm thế nào để chương trình chỉ đọc lên những dữ liệu tf-idf cần thiết trong block. Giải pháp được đưa ra là định dạng lại cách lưu trữ trọng số tf-idf trên mỗi block, mỗi record sẽ bao gồm hai trường, trường thứ nhất là một từ, trường thứ hai gồm một danh sách các tài liệu có chứa từ trong trường thứ nhất kèm theo trọng số tf-idf của từ đó tương ứng trong mỗi tài liệu. Như vậy mỗi record sẽ có dạng: <w, {d1:tf-idf$_{w\ d1}$, d2:tf-idf$_{w\ d2}$, …, dn:tf-idf$_{w\ dn}$}>. Nhờ đó, chương trình sẽ chỉ đọc lên những record có giá trị của trường thứ nhất xuất hiện trong biểu thức tìm kiếm.

Ngoài ra, xét tử số của công thức tính độ tương đồng giữa tài liệu d với biểu thức tìm kiếm q:

$$\vec{V}(d).\vec{V}(q)$$

$$= tf-idf_{new\ \ q} \cdot tf-idf_{new\ \ d} + tf-idf_{york\ \ q} \cdot tf-idf_{york\ \ d}$$

$$= tf_{new\ \ q} \cdot tf_{new\ \ d} \cdot (idf_{new})^2 + tf_{york\ \ q} \cdot tf_{york\ \ d} \cdot (idf_{york})^2$$

Ta thấy, trên mỗi block nếu thay vì lưu trữ trọng số tf-idf = tf.idf như bình thường, ta lưu tf-idf = tf.(idf $^2$) thì khi tính độ tương đồng giữa các tài liệu với biểu thức tìm kiếm, ngoài dữ liệu được đọc lên từ block, ta chỉ cần có thêm chỉ số tf của

các từ trong biểu thức tìm kiếm là đã có đủ cơ sở để tính toán. Đều này sẽ giúp quá trình xử lý biểu thức tìm kiếm diễn ra nhanh hơn.

# CHƯƠNG 4: KẾT QUẢ THỰC HIỆN

Với mục tiêu và phạm vi nghiên cứu của mình, trong phần này tôi chỉ tập trung trình bày kết quả thực hiện, kết luận và đề xuất cho các nội dung được phân công thực hiện.

## 4.1. KẾT QUẢ THỰC HIỆN

### 4.1.1. Phần nghiên cứu mô hình MapReduce và nền tảng Hadoop

Sau quá trình nghiên cứu tài liệu, nhóm chúng tôi nói chung và bản thân tôi nói riêng, đã nắm được những kiến thức cơ bản về mô hình MapReduce và nền tảng Hadoop. Để kiểm nghiệm kết quả nghiên cứu tài liệu và tạo cơ sở triển khai công cụ tìm kiếm, nhóm chúng tôi đã cài đặt thử nghiệm một Hadoop cluster tại phòng 3.2, khoa Công nghệ thông tin và truyền thông, trường Đại học Cần Thơ, kết quả cài đặt thử nghiệm rất thành công. Hadoop cluster được cài đặt bao gồm 18 máy, trong đó có 03 máy master và 15 máy slave. Chi tiết như sau:

- Một máy master chạy JobTracker.
- Một máy master chạy NameNode.
- Một máy master chạy SecondaryNameNode.
- 15 máy slave chạy các DataNode và TaskTracker.

Ngoài ra, tại phòng máy còn có thêm một máy được cài đặt web server để chạy giao diện web của hệ thống tìm kiếm.

### 4.1.2. Phần xây dựng hệ thống xử lý yêu cầu tìm kiếm

Để đánh giá hệ thống tìm kiếm đã được xây dựng, nhóm chúng tôi thực hiện triển khai hệ thống lên Hadoop cluster đã được cài đặt trước đó tại phòng 3.2, khoa Công nghệ thông tin và truyền thông, trường Đại học Cần Thơ, sau đó tiến hành thực nghiệm, và nhận được kết quả rất khả quang. Phần xử lý yêu cầu tìm kiếm hoạt động khá ổn định và chính xác, hiệu năng làm việc rất cao.

| Kích cở tập tài liệu (file) | Thời gian xử lý yêu cầu tìm kiếm trung bình (giây) |
|---|---|
| 50 | 0.017125 |
| 500 | 0.0295 |
| 1000 | 0.0407142857 |
| 2500 | 0.1197 |
| 5500 | 0.2605 |

Thời gian xử lý yêu cầu tìm kiếm trung bình trên mỗi tập tài liệu

Sau đây là chi tiết kết quả thực nghiệm của phần hệ thống xử lý yêu cầu tìm kiếm.

**Xử lý biểu thức tìm kiếm**

Biểu thức tìm kiếm có độ dài tương đối ngắn, quá trình xử lý biểu thức tìm kiếm diễn ra trong thời gian rất nhanh. Trong quá trình thực nghiệm, với các biểu thức tìm kiếm có độ dài không quá 10 từ, thì thời gian xử lý không quá 0.01 giây. Tuy nhiên, trong một vài trường hợp, quá trình xử lý tách từ cho biểu thức tìm kiếm bị treo, nguyên nhân sẽ được trình bày cụ thể trong phần kết luận và đề xuất.

**Tính độ tương đồng**

Đây là giai đoạn được đánh giá là tốn nhiều thời gian thực hiện nhất trong quá trình xử lý yêu cầu tìm kiếm. Như đã được trình bày ở trên, nhóm chúng tôi đã có những cải tiến cho phần này nhằm tăng tốc độ làm việc của hệ thống. Một trong những cải tiến quan trọng nhất là việc xây dựng và sử dụng một MapReduce layer mới thay vì sử dụng MapReduce layer của nền tảng Hadoop. Với sự cải tiến này, thời gian thực hiện của giai đoạn tính độ tương đồng đã tăng lên gấp nhiều lần so với hệ thống chưa được cải tiến. Tuy nhiên, MapReduce layer mới do được xây dựng trong thời gian ngắn, chưa có thời gian thực nghiệm, sửa lỗi đủ lâu, nên khả năng xử lý ngoại lệ chưa thật sự tốt.

| Kích cở tập tài liệu (file) | Thời gian tính độ tương đồng trung bình (giây) | |
|---|---|---|
| | Hệ thống chưa cải tiến | Hệ thống cải tiến |
| 50 | 83.29 | 0.0162 |
| 500 | 83.74 | 0.0265 |
| 1000 | 84.2233333333 | 0.0311428571 |
| 2500 | 84.6025 | 0.0626 |
| 5500 | 85.94 | 0.1125 |

Bảng so sánh thời gian tính độ tương đồng trung bình

**Sắp xếp tài liệu theo độ tương đồng và xuất kết quả**

Giải thuật sắp xếp được sử dụng trong giai đoạn này là giải thuật Quick sort, qua thực nghiệm trên hệ thống thật cho thấy, thời gian sắp xếp là rất nhanh. Thời gian sắp xếp lâu nhất đo được khi thực hiện tìm kiếm trên tập tài liệu 5500 file là 0.312 giây - khi sắp xếp cho 5331 tài liệu. Thời gian sắp xếp trung bình khi thực nghiệm trên tập tài liệu 5500 file là 0.148 giây. Ngoài ra, với cải tiến trong quá trình xuất kết quả đã được trình bày ở trên, trong hầu hết các trường hợp, người đưa ra yêu cầu tìm kiếm sẽ nhận trước một phần kết quả, chớ không cần phải đợi toàn bộ quá trình sắp xếp được thực hiện xong, đều này cũng làm tăng đáng kể tốc độ đáp trả yêu cầu tìm kiếm của hệ thống.

## 4.2. KẾT LUẬN VÀ ĐỀ XUẤT

### 4.2.1. Phần nghiên cứu mô hình MapReduce và nền tảng Hadoop

Sau toàn bộ quá trình thực hiện đề tài, bản thân tôi đã nắm được những kiến thức cơ bản về mô hình MapReduce và nền tảng Hadoop. Đủ khả năng cài đặt, cấu hình và vận hành một Hadoop cluster. Tuy nhiên, với các tùy chọn nâng cao nhằm tăng hiệu xuất của hệ thống, như thiết lập số map task, reduce task trên mỗi máy slave; thiết lập đa luồng cho map task, v.v. bản thân vẫn còn nhiều vấn đề chưa thông suốt. Trong thời gian tới, tôi sẽ tiếp tục nghiên cứu hệ thống, giải quyết các vấn đề còn tồn đọng, cũng như những vấn đề phát sinh thêm.

### 4.2.2. Phần xây dựng hệ thống xử lý yêu cầu tìm kiếm

Hệ thống xử lý yêu cầu tìm kiếm hoạt động khá ổn định và hiệu quả, hiệu năng làm việc cao. Kết quả ghép nối với hệ thống lập chỉ mục rất tốt, hai hệ thống hoạt động rất đồng bộ. Tuy nhiên, hệ thống xử lý yêu cầu tìm kiếm được xây dựng vẫn còn tồn tại một số hạn chế chưa được giải quyết, sau đây là chi tiết về các hạn chế và hướng giải quyết cụ thể:

- Quá trình xử lý biểu thức tìm kiếm đôi khi bị treo, nguyên nhân phát sinh từ gói VNTokenizer được sử dụng trong giai đoạn tách từ. Tuy vấn đề trên rất hiếm khi xảy ra nhưng mỗi khi gặp phải, hệ thống sẽ không thể đáp ứng được yêu cầu tìm kiếm của người dùng. Có hai hướng giải quyết cho vấn đề vừa nêu: Hướng thứ nhất là, tiếp tục nghiên cứu, cải tiến VNTokenizer (VNTokenizer là một thư viện Java, nguồn mở); Hướng thứ hai là, nghiên cứu, tìm ra một giải pháp tách từ mới, hoạt động ổng định hơn giải pháp sử dụng VNTokenizer.

- MapReduce layer mới được xây dựng trong quá trình cải tiến hoạt động chưa thật sự ổn định, dữ liệu được gửi từ các Search server về Search client đôi khi bị sai lệch và thất thoát. Để giải quyết vấn đề này cần phải thiết kế, bổ sung các giải thuật nhận biết và xử lý ngoại lệ cho MapReduce layer mới.

- Hai quá trình xử lý biểu thức tìm kiếm, sắp xếp tài liệu theo độ tương đồng và xuất kết quả, chỉ được ứng dụng kỷ thuật đa luồng để cài đặt. Để hệ thống đáp ứng được lượng người dùng lớn, cần áp dụng kỹ thuật xử lý phân tán song song cho hai quá trình này.

# PHỤ LỤC

## PHỤ LỤC 1: HƯỚNG DẪN CÀI ĐẶT, VẬN HÀNH HADOOP CLUSTER

### 1.1. Những phiên bản phần mền, hệ điều hành dùng cài đặt trong phần hướng dẫn

- Hệ điều hành Ubuntu Desktop 12.04
- Java phiên bản 1.6
- SSH phiên bản 6.0 cài sẵn trên hệ điều hành
- Hadoop phiên bản 1.0.4

### 1.2. Hướng dẫn cài đặt

Sau đây là hướng dẫn cài đặt một Hadoop cluster, lên một hệ thống các máy tính được nối mạng với nhau và cài đặt sẵn hệ điều hành Ubuntu Desktop 12.04. Giải sử, hệ thống giành cho cài đặt gồm có n máy tính, ta chọn ra các máy chạy JobTracker, NameNode, SercondaryNameNode và slave như sau:

- Host_1: Máy master chạy JobTracker
- Host_2: Máy master chạy NameNode
- Host_3: Máy master chạy SecondaryNameNode
- Host_4 đến Host_n: Là các máy slave

Chú ý: Các máy master và slave có thể trùng nhau, có nghĩa là một máy tính có thể vừa đóng vai trò là master chạy JobTracker, vừa là máy slave, hoặc vừa là máy master chạy JobTracker vừa là master chạy NameNode, v.v. Trong trường hợp đặc biệt, có thể cài Hadoop cluster trên một máy tính duy nhất.

**Những cài đặt, thiết lập chung cho tất cả các máy trên hệ thống:**

- Cập nhật tập tin hosts

| Thay thế các phần được gạch chân trong nội dung bên dưới bằng thông tin tương ứng của hệ thống, sau đó thêm vào cuối tập tin /etc/hosts - Mỗi dòng là thông tin về HostIP và HostName của mỗi máy trong hệ thống |
|---|
| Host_1-IP Host_1-Name<br>Host_2-IP Host_2-Name<br>…<br>Host_n-IP Host_n-Name |

- Cài đặt Java: Hadoop yêu cầu các máy tính trên cluster phải được cài Java với phiên bản thấp nhất là 1.5, và phiên bản khuyên dùng là 1.6. Do đó, để hệ thống hoạt động một cách tốt nhất, phiên bản Java 1.6 sẽ được hướng dẫn cài đặt. Ngoài ra, trong hướng dẫn sau, việc cài đặt Java sẽ được thực hiện thông qua một nhà phân phối, nếu thấy đó là không an toàn, bạn có thể tải và cài đặt Java theo hướng dẫn tại trang chủ của Oracle.

| Thực hiện các lệnh sau trong Terminal |
|---|
| # Thêm kho chứa – repository cho Ubuntu<br>$ sudo add-apt-repository ppa:ferramroberto/java<br># Cập nhật lại danh sách kho chứa<br>$ sudo apt-get update<br># Cài đặt Java 6 JDK - Java 1.6<br>$ sudo apt-get install sun-java6-jdk<br># Cập nhật phiên bản Java mặc định<br>$ sudo update-java-alternatives -s java-6-sun |

- Tạo người dùng riêng cho Hadoop:

| Thực hiện các lệnh sau trong Terminal |
|---|
| # Thêm nhóm người dùng hadoop<br>$ sudo addgroup hadoop<br># Thêm người dùng hduser thuộc nhóm người dùng hadoop |

```
$ sudo adduser --ingroup hadoop hduser
```

- Cấu hình SSH:

Thực hiện các lệnh sau trong Terminal

```
# Đăng nhập vào tài khoảng người dùng hduser
$ su - hduser
# Tạo ra khóa chứng thực SSH cho tài khoảng hduser
$ ssh-keygen -t rsa -P ""
```

- Cài đặt và cấu hình Hadoop

Tải hadoop phiên bản 1.0.4 tại website http://hadoop.apache.org/, sau đó thực hiện các lệnh sau trong Terminal để cài đặt Hadoop

```
# Giải nén gói hadoop
$ sudo tar xzf hadoop-1.0.4.tar.gz
# Di chuyển thư mục cài đặt hadoop về thư mục /usr/local
$ sudo mv hadoop-1.0.4 /usr/local/hadoop
# Cập nhật owner cho thư mục cài đặt hadoop
$ sudo chown -R hduser:hadoop hadoop
```

Chèn nội dung sau vào cuối tập tin /home/hduser/.bashrc để thêm các thiết lập đường dẫn cho người dùng hduser

```
# Set Hadoop-related environment variables
export HADOOP_HOME=/usr/local/hadoop
# Set JAVA_HOME
export JAVA_HOME=/usr/lib/jvm/java-6-sun
unalias fs &> /dev/null
alias fs="hadoop fs"
unalias hls &> /dev/null
alias hls="fs -ls"
# $ lzohead /hdfs/path/to/lzop/compressed/file.lzo
# Requires installed 'lzop' command.
lzohead () {
    hadoop fs -cat $1 | lzop -dc | head -1000 | less
}
```

```
# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin
```

Chèn nội dung sau vào cuối tập tin hadoop/conf/hadoop-env.sh để thêm thiết lập đường dẫn Java cho Hadoop

```
# The java implementation to use.  Required.
export JAVA_HOME=/usr/lib/jvm/java-6-sun
```

Thực hiện các lệnh sau trong Terminal để tạo thư mục lưu trữ cho HDFS

```
# Tạo thư mục lưu trữ cho HDFS
$ sudo mkdir -p /app/hadoop/tmp
# Cập nhật owner cho thư mục tmp
$ sudo chown hduser:hadoop /app/hadoop/tmp
# Cập nhận quyền cho thực mục tmp
$ sudo chmod 750 /app/hadoop/tmp
```

Thay thế các phần được gạch chân trong nội dung bên dưới bằng thông tin tương ứng của hệ thống, sau đó thêm vào giữa 2 thẻ <configuration> và </configuration> trong tập tin hadoop/conf/core-site.xml

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
</property>

<property>
  <name>fs.default.name</name>
  <value>hdfs://Host_2-Name:9000</value>
</property>
```

Thay thế các phần được gạch chân trong nội dung bên dưới bằng thông tin tương ứng của hệ thống, sau đó thêm vào giữa 2 thẻ <configuration> và </configuration> trong tập tin hadoop/conf/mapred-site.xml

```
<property>
  <name>mapred.job.tracker</name>
  <value>Host_1-Name:9001</value>
</property>
<property>
  <name>mapred.system.dir</name>
  <value>/app/hadoop/tmp/mapred/system</value>
</property>
```

Thay thế các phần được gạch chân trong nội dung bên dưới bằng thông tin tương ứng của hệ thống, sau đó thêm vào giữa 2 thẻ <configuration> và </configuration> trong tập tin hadoop/conf/hdfs-site.xml

```
<property>
  <name>dfs.replication</name>
  <value>2</value>
</property>
<property>
  <name>dfs.name.dir</name>
  <value>/app/hadoop/dfs/name</value>
</property>
<property>
  <name>dfs.data.dir</name>
  <value>/app/hadoop/dfs/data</value>
</property>
<property>
  <name>dfs.http.address</name>
  <value>Host_2-Name:9002</value>
</property>
```

**Những cài đặt, thiết lập riêng trên (các) máy master:**

- Cấu hình SSH:

| Thay thế các phần được gạch chân trong các lệnh bên dưới bằng thông tin tương ứng của hệ thống, sau đó thực hiện trong Terminal để lấy về khóa chứng thực SSH của người dùng hduser trên các máy trong hệ thống. |
|---|
| # Đăng nhập vào tài khoảng người dùng hduser<br>$ su – hduser<br># Lấy khóa chứng thực SSH<br>$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@Host_1-Name<br>$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@Host_2-Name<br>…<br>$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@Host_n-Name<br># Kích hoạt khóa<br>$ ssh Host_1-Name<br>$ ssh Host_2-Name<br>…<br>$ ssh Host_n-Name |

| Thay thế các phần được gạch chân trong nội dung bên dưới bằng thông tin tương ứng của hệ thống, sau đó ghi đè lên nội dung trong tập tin hadoop/conf/masters của máy master chạy NameNode |
|---|
| # Tên của các máy master chạy SecondaryNameNode<br>Host_3-Name |

| Thay thế các phần được gạch chân trong nội dung bên dưới bằng thông tin tương ứng của hệ thống, sau đó ghi đè lên nội dung trong tập tin hadoop/conf/slaves của máy master chạy JobTracker |
|---|
| # Tên của các máy slave<br>Host_4-Name<br>Host_5-Name<br>…<br>Host_n-Name |

| Thực hiện các lệnh sau trong Tereminal trên máy master chạy NameNode để format NameNode - đều này cần phải được thực hiện trước lần khởi động Hadoop cluster đầu tiên. |
| --- |
| # Đăng nhập vào tài khoảng người dùng hduser<br>$ su – hduser<br># Format NameNode<br>$ /usr/local/hadoop/bin/hadoop namenode -format |

## 1.3. Hướng dẫn vận hành

### Khởi động hệ thống

| Thực hiện các lệnh sau trong Terminal trên máy master chạy NameNode để khởi động hệ thống HDFS. |
| --- |
| # Đăng nhập vào tài khoảng người dùng hduser<br>$ su – hduser<br># Start HSFD<br>$ /usr/local/hadoop/bin/hadoop/bin/start-dfs.sh |

| Thực hiện các lệnh sau trong Terminal trên máy master chạy JobTracker để khởi động hệ thống MapReduce. |
| --- |
| # Đăng nhập vào tài khoảng người dùng hduser<br>$ su – hduser<br># Start mapred layer<br>$ /usr/local/hadoop/bin/hadoop/bin/start-mapred.sh |

### Tắt hệ thống

| Thực hiện các lệnh sau trong Terminal trên máy master chạy NameNode để tắt hệ thống HDFS. |
| --- |
| # Đăng nhập vào tài khoảng người dùng hduser<br>$ su – hduser<br># Stop HDFS<br>$ /usr/local/hadoop/bin/hadoop/bin/stop-dfs.sh |

| Thực hiện các lệnh sau trong Terminal trên máy master chạy JobTracker để tắt hệ thống MapReduce. |
|---|
| # Đăng nhập vào tài khoảng người dùng hduser<br>$ su – hduser<br># Stop mapred layer<br>$ /usr/local/hadoop/bin/hadoop/bin/stop-mapred.sh |

Ngoài ra, có thể tham khảo hướng dẫn cài đặt và các lệnh thông dụng của Hadoop trên trang chủ của nền tảng này, địa chỉ: http://hadoop.apache.org/.

# PHỤ LỤC 2: HƯỚNG DẪN VẬN HÀNH, SỬ DỤNG HỆ THỐNG TÌM KIẾM

## 2.1. Hướng dẫn vận hành

Để hệ thống khởi động, cần thực hiện lần lược các thao tác sau:

- Tạo hệ thống thư mục để lưu trử dữ liệu trên HDFS và máy chạy web server.
- Khởi động Hadoop.
- Khởi động module Master.jar tại máy master chạy JobTracker.
- Khởi động các module ListenQuery.jar và CheckUpload.jar tại máy chạy web server.

Chi tiết việc thực hiện các bước để khởi động hệ thống khá phức tạp, để đơn giản hóa việc khởi động hệ thống, chúng tôi có viết một Linux Shell Script kèm theo mã nguồn của chương trình.

Để tắt một module đang hoạt động, chỉ cần ấn phìm bất kỳ trong giao diện dòng lệnh chạy module đó.

## 2.2. Hướng dẫn sử dụng

Để thao tác với hệ thống, người dùng truy cập vào địa chỉ của web server.

Thông qua giao diện web, người dùng dể dàng upload tài liệu mới, hoặc truyền yêu cầu tìm kiếm của mình vào hệ thống.

# TÀI LIỆU THAM KHẢO

[1] **Apache**; *Hadoop 1.0.4 Documentation*; http://hadoop.apache.org/docs/r1.0.4/index.html.

[2] **Lê Hồng Phương**; *vnTokenizer Document*; http://mim.hus.vnu.edu.vn/phuonglh/softwares/vnTokenizer.

[3] **Tom White**; *Hadoop The Definitive Guide 3rd Edition*.

[4] **Nguyễn Minh Thuận, Nguyễn Trọng Thức**; *Nghiên cứu nền tảng tính toán song song với MapReduce và Hadoop. Áp dụng cho việc xây dựng wordnet tiếng Việt và tạo chỉ mục tài liệu*; Cần Thơ, 2012.

[5] **Bùi Thị Hồng Phúc**; *Xây dựng một ứng dụng minh họa cho khả năng của MongdoDB*; Cần Thơ, 2012.

[6] **G. Salton, A. Wong, C. S. Yang**; *A Vector Space Model for Automatic Indexing*.