BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM

NGUYỄN THỊ BÍCH THÙY

NGHIÊN CỨU MÔ HÌNH KIỂM SOÁT TRUY XUẤT CHO DỮ LIỆU LỚN

LUẬN VĂN THẠC SỸ

Chuyên ngành: CÔNG NGHỆ THÔNG TIN

Mã ngành: 60480201

TP. HCM, tháng 10/2015

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM

NGUYỄN THỊ BÍCH THÙY

NGHIÊN CỨU MÔ HÌNH KIỂM SOÁT TRUY XUẤT CHO DỮ LIỆU LỚN

LUẬN VĂN THẠC SỸ

Chuyên ngành: CÔNG NGHỆ THÔNG TIN

Mã ngành: 60480201

CÁN BỘ HƯỚNG DẪN KHOA HỌC: TS. VÕ ĐÌNH BẢY

TP. HCM, tháng 10/2015

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH TẠI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM

Cán bộ hướng dẫn khoa học : **TS.VÕ ĐÌNH BẢY**

Luận văn Thạc sĩ được bảo vệ tại Trường Đại học Công nghệ TP. HCM ngày 17 tháng 10 năm 2015.

Thành phần Hội đồng đánh giá Luận văn Thạc sĩ gồm:

TT	Họ và tên	Chức danh Hội đồng
1	PGS.TS. Lê Hoài Bắc	Chủ tịch
2	GS.TSKH. Hoàng Văn Kiếm	Phản biện 1
3	TS.Vũ Thanh Hiền	Phản biện 2
4	TS. Hồ Đắc Nghĩa	Ủy viên
5	TS. Cao Tùng Anh	Ủy viên, Thư ký

Xác nhận của Chủ tịch Hội đồng đánh giá Luận sau khi Luận văn đã được sửa chữa.

Chủ tịch Hội đồng đánh giá LV

PGS.TS. Lê Hoài Bắc

TRƯỜNG ĐH CÔNG NGHỆ TP. HCM
PHÒNG QLKH – ĐTSĐH

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

TP. HCM, ngày..... tháng….. năm 20.....

NHIỆM VỤ LUẬN VĂN THẠC SĨ

Họ tên học viên: Nguyễn Thị Bích Thùy.....Giới tính: Nữ
Ngày, tháng, năm sinh: 26/05/1980.....Nơi sinh: TP.HCM
Chuyên ngành: Công nghệ thông tin.....MSHV: 1241860022

I- Tên đề tài:

Nghiên cứu mô hình kiểm soát truy xuất cho dữ liệu lớn

II- Nhiệm vụ và nội dung:

- Nghiên cứu về dữ liệu lớn
- Nghiên cứu mô hình kiểm soát truy xuất dữ liệu
- Nghiên cứu mô hình kiểm soát truy xuất cho dữ liệu lớn.
- Nghiên cứu thực nghiệm ứng dụng kiểm soát truy xuất cho dữ liệu lớn.

III- Ngày giao nhiệm vụ: *08/03/2015*

IV- Ngày hoàn thành nhiệm vụ: *08/09/2015*

V- Cán bộ hướng dẫn: *TS.Võ Đình Bảy.*

CÁN BỘ HƯỚNG DẪN
(Họ tên và chữ ký)

TS.Võ Đình Bảy

KHOA QUẢN LÝ CHUYÊN NGÀNH
(Họ tên và chữ ký)

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi. Các số liệu, kết quả nêu trong Luận văn là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Tôi xin cam đoan rằng mọi sự giúp đỡ cho việc thực hiện Luận văn này đã được cảm ơn và các thông tin trích dẫn trong Luận văn đã được chỉ rõ nguồn gốc.

Học viên thực hiện Luận văn

Nguyễn Thị Bích Thùy

LỜI CÁM ƠN

Trong suốt quá trình học tập, nghiên cứu và hoàn thành luận văn tốt nghiệp, tôi đã nhận được sự hướng dẫn, giúp đỡ và động viên rất quý báu của Thầy Cô, Gia đình, Bạn bè và anh chị đồng nghiệp.

Trước hết, tôi xin bày tỏ lòng kính trọng và tri ân sâu sắc đến TS.Võ Đình Bảy, người hướng dẫn khoa học đã tận tâm giúp đỡ, dạy bảo và động viên tôi. Tôi cũng xin gửi lời cảm ơn sâu sắc đến Quý thầy cô đã tận tình dìu dắt, truyền dạy kiến thức cho tôi trong thời gian vừa qua.

Xin cảm ơn Quý thầy cô Ban giám hiệu, Phòng Quản lý khoa học – Đào tạo Sau đại học, Khoa Công nghệ thông tin trường Đại học Công Nghệ Thành phố Hồ Chí Minh đã tạo điều kiện thuận lợi trong thời gian tôi học tập tại trường.

Sau cùng, xin cảm ơn gia đình, bạn bè đã động viên, khích lệ tôi trong suốt quá trình học tập và thực hiện luận văn.

Dù đã có nhiều cố gắng trong quá trình thực hiện luận văn tốt nghiệp, song chắc chắn rằng luận văn sẽ không thể tránh khỏi thiếu sót. Tôi rất mong nhận được sự góp ý của quý thầy cô, anh chị em đồng nghiệp và các bạn.

Tp.Hồ Chí Minh, tháng 10 năm 2015

Nguyễn Thị Bích Thùy

TÓM TẮT

Hệ thống điều khiển truy xuất Access Control (AC) là một trong những thành phần quan trọng nhất về an ninh mạng; là kỹ thuật cho phép kiểm soát việc truy nhập đến một tài nguyên tính toán cho một người dùng hoặc một nhóm người dùng nào đó. Điều khiển truy cập thường được sử dụng như lớp phòng vệ thứ nhất, nhằm ngăn chặn các các phần mềm độc hại và các hành động tấn công, đột nhập vào các hệ thống máy tính và mạng, hoặc truy cập trái phép vào dữ liệu và các tài nguyên tính toán. Vấn đề này ngày càng trở nên nghiêm trọng phức tạp hơn trong các hệ thống phần mềm, chẳng hạn như hệ thống xử lý Big Data (BD), đó là hệ thống được triển khai để quản lý một số lượng lớn các thông tin và nguồn tài nguyên được tổ chức thành một cụm xử lý Big Data (BD).

Về cơ bản, kiểm soát truy cập BD đòi hỏi sự phối hợp xử lý để được bảo vệ như hệ thống điện toán đám mây dựa trên nền tảng tính toán cho quản lý kiểm soát truy cập phân tán. Trong điều kiện hạ tầng mạng cũng như nhân lực quản trị hệ thống của các cơ quan, tổ chức ở Việt Nam hiện nay còn hạn chế, việc nghiên cứu về điều khiển truy cập cho dữ liệu lớn BD để tìm giải pháp ứng dụng phù hợp là thực sự cần thiết. Luận văn này trình bày nghiên cứu về dữ liệu lớn, mô hình kiểm soát truy xuất cho dữ liệu, đề xuất mô hình kiểm soát truy xuất cho dữ liệu lớn và thực nghiệm một ý tưởng AC phân cụm xử lý BD.

ABSTRACT

Access control systems Access Control (AC) is one of the most important components network security; a technique that allows to control the access to a computing resource to a user or a particular group of users. Access controls are often used as the first layer of defense, preventing the malicious software and the strike action, break into computer systems and networks, or unauthorized access to data and the computing resources. This problem is becoming more serious in the more complex software systems, such as Big Data processing systems (BD), which is the system to be deployed to manage a large amount of information and resources are organized in a cluster handle Big Data (BD).

Basically, control access to BD requires coordination processor to be protected as the system based on cloud computing platform for managing distributed access control. In terms of network infrastructure and system management personnel of the agencies and organizations in Vietnam today is still limited, the study of access control for large data BD to find solutions consistent application where is really necessary. This thesis presents a research on big data, model control access to data, the proposed model to control access for big data and applied idea for clustering processing AC BD.

MỤC LỤC

Danh mục các biểu đồ, đồ thị, sơ đồ, hình ảnh

Chương 1: Mở đầu

Sự bùng nổ của các dịch vụ trực tuyến cùng sự phát triển không ngừng của công nghệ và các thiết bị di động đã làm gia tăng nhu cầu quản lý và chia sẻ thông tin, đặc biệt là trong các hệ thống quản lý giáo dục, y tế, giải trí,…, các phần mềm ứng dụng cho các cơ quan quản lý nhà nước nhằm đáp ứng yêu cầu quản lý, thống kê, dự báo, hoạch định,…. Các thông tin này được lưu trữ với số lượng dữ liệu lớn, dưới nhiều dạng khác nhau cũng như tốc độ sinh ra nhanh, các dữ liệu này được gọi là dữ liệu lớn. Số lượng dữ liệu càng tăng và đa dạng kéo theo việc bảo mật các dữ liệu trở nên cấp thiết và khó khăn hơn. Do đó, bảo mật dữ liệu lớn được xem là một trong những thách thức quan trọng đặt ra cho nghiên cứu về dữ liệu lớn và các ứng dụng liên quan.

Dữ liệu lớn ngày càng thu hút sự quan tâm của các nhà nghiên cứu về khía cạnh bảo mật. Có 3 vấn đề quan trọng trong việc bảo vệ tính riêng tư cho dữ liệu lớn: điều khiển truy xuất (Access control), kiểm tra (auditing), bảo mật thống kê (statistical privacy). Trong đó access control (kiểm soát truy xuất) là vấn đề cần thiết trong việc bảo vệ dữ liệu khỏi những truy xuất trái phép, giúp cho việc quản lý và chia sẻ dữ liệu hiệu quả hơn. Đây cũng là vấn đề trọng tâm được quan tâm trong đề tài này.

Đề tài này nhằm nghiên cứu về dữ liệu lớn trong tình trạng bùng nổ dữ liệu nói chung, đã và đang đòi hỏi một giải pháp kiểm soát truy xuất chặt chẽ để bảo vệ dữ liệu tránh khỏi những truy xuất không hợp lệ nhằm tăng tính an toàn cho dữ liệu, tăng độ tin cậy dữ liệu cho các ứng dụng liên quan.

Luận văn gồm 6 chương với nội dung như sau:

Chương 1- Mở đầu .

Chương 2- Tổng quan về dữ liệu lớn.

Chương 3- Mô hình điều khiển truy cập dữ liệu.

Các biện pháp điều khiển truy cập thông dụng đi sâu phân tích 4 cơ chế điều khiển truy cập phổ biến là điều khiển truy cập tùy quyền (DAC),

điều khiển truy cập bắt buộc (MAC), điều khiển truy cập dựa trên vai trò (Role-Based AC) và điều khiển truy cập dựa trên luật (Rule-Based AC).

Chương 4- Điều khiển truy xuất cho dữ liệu lớn

Chương 5- Thực nghiệm và kết quả

Chương 6- Kết luận và hướng phát triển.

Chương 2: Tổng quan về dữ liệu lớn

2.1 Giới thiệu

Hiện đã có rất nhiều thảo luận về khái niệm Big Data (Dữ liệu lớn), nhưng Big Data đơn giản là dữ liệu tiêu chuẩn thường được phân phối qua nhiều địa điểm, từ đa dạng các nguồn tin, trong các định dạng khác nhau và thường không có cấu trúc nhất định. Những thách thức đầu tiên đối với Big Data là khả năng quản lý khối lượng và đảm bảo truy cập thường xuyên. Bởi vì, bảo vệ dữ liệu khỏi sự xâm nhập và phá hoại, đồng thời duy trì truy cập an toàn chính là ưu tiên hàng đầu cho các chuyên gia bảo mật hiện nay.

2.2 Định nghĩa và các đặc trưng

2.2.1 Big Data là gì?

Big Data là thuật ngữ dùng để chỉ một tập hợp dữ liệu rất lớn và phức tạp đến nỗi những công cụ, ứng dụng xử lý dữ liệu truyền thống không thể đảm đương được. Kích cỡ của Big Data đang tăng lên từng ngày, tính đến năm 2012 nó đã lên hàng exabyte (1 exabyte = 1 tỷ gigabyte). Các nhà khoa học thường xuyên gặp phải những hạn chế do tập dữ liệu lớn trong nhiều lĩnh vực, như khí tượng học, di truyền học, mô phỏng vật lý phức tạp, nghiên cứu sinh học và môi trường. Những hạn chế cũng ảnh hưởng đến việc tìm kiếm trên internet, tài chính và thông tin kinh doanh.[24]

Theo IBM, lượng thông tin công nghệ bình quân đầu người trên thế giới tăng gần gấp đôi mỗi 40 tháng kể từ năm 1980. Tính đến năm 2012, mỗi ngày có 2,5 exabyte dữ liệu được tạo ra. Còn theo tài liệu của Intel vào tháng 9-2013, hiện nay thế giới đang tạo ra 1 petabyte (1 petabyte = 1.000 terabyte) dữ liệu trong mỗi 11 giây (tương đương một đoạn video HD dài 13 năm). [12]

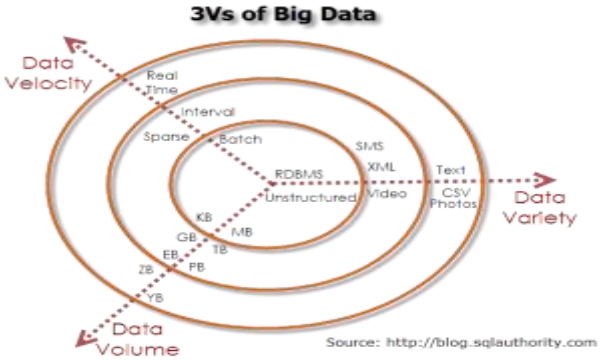
Bản thân các công ty, doanh nghiệp cũng đang sở hữu Big Data của riêng mình, chẳng hạn trang bán hàng trực tuyến eBay sử dụng 2 trung tâm dữ liệu với dung lượng lên đến 40 petabyte để chứa những truy vấn, tìm kiếm, đề xuất cho khách hàng cũng như thông tin về hàng hóa của mình. Nhà bán lẻ online Amazon.com đã sử dụng một hệ thống Linux để xử lý hàng triệu hoạt động mỗi ngày cùng những yêu cầu từ khoảng nửa triệu đối tác bán hàng. Tính đến năm 2005, họ từng sở hữu 3 cơ

sở dữ liệu Linux lớn nhất thế giới với dung lượng 7,8TB, 18,5TB và 24,7TB. Tương tự, Facebook cũng phải quản lý 50 tỷ bức ảnh từ người dùng tải lên, trong khi YouTube hay Google phải lưu lại hết các lượt truy vấn và video của người dùng cùng nhiều loại thông tin khác có liên quan.

Năm 2011, Tập đoàn McKinsey đề xuất những công nghệ có thể dùng với Big Data, bao gồm crowsourcing (tận dụng nguồn lực từ nhiều thiết bị điện toán trên toàn cầu để cùng xử lý dữ liệu), các thuật toán về gen và di truyền, những biện pháp machine learning (các hệ thống có khả năng học hỏi từ dữ liệu - một nhánh của trí tuệ nhân tạo), xử lý ngôn ngữ tự nhiên (giống như Siri hay Google Voice Search, nhưng cao cấp hơn), xử lý tín hiệu, mô phỏng, phân tích chuỗi thời gian, mô hình hóa, kết hợp các server mạnh lại với nhau...

Ngoài ra, các cơ sở dữ liệu hỗ trợ xử lý dữ liệu song song, ứng dụng hoạt động dựa trên hoạt động tìm kiếm, tập tin hệ thống (file system) dạng rời rạc, các hệ thống điện toán đám mây (bao gồm ứng dụng, nguồn lực tính toán cũng như không gian lưu trữ) và bản thân internet cũng là những công cụ đắc lực phục vụ cho công tác nghiên cứu và trích xuất thông tin từ Big Data. Hiện nay cũng có vài cơ sở dữ liệu theo dạng quan hệ (bảng) có khả năng chứa hàng petabyte dữ liệu, chúng cũng có thể tải, quản lý, sao lưu và tối ưu hóa cách sử dụng Big Data [24].

Theo http://blog.SQLAuthority.com, mô hình 3V để định nghĩa Big Data là là khối lượng (volume), vận tốc (velocity) và chủng loại (variety).



Hình 2.2 Mô hình 3V[5]

Volume (Khối lượng)

Việc lưu trữ khối lượng dữ liệu đang tăng trưởng theo cấp số nhân chứ không chỉ đơn thuần là dữ liệu văn bản. Chúng ta có thể tìm thấy dữ liệu trong các định dạng phim (video), nhạc (music), hình ảnh (image) lớn trên các kênh truyền thông xã hội. Khối lượng dữ liệu ngày nay có thể lên đến hàng Terabyte và Petabyte. Khối lượng dữ liệu ngày càng phát triển thì các ứng dụng và kiến trúc xây dựng để hỗ trợ dữ liệu cần phải được đánh giá lại khá thường xuyên. Khối lượng lớn dữ liệu thực sự đại diện cho big data.[5]

Velocity (Vận tốc)

Sự tăng trưởng dữ liệu và các phương tiện truyền thông xã hội đã thay đổi cách chúng ta nhìn vào dữ liệu. Ngày nay, mọi người trả lời trên kênh truyền thông xã hội để cập nhật những diễn biến mới nhất. Trên phương tiện truyền thông xã hội đôi khi các thông báo cách đó vài giây (tweet, status,….) đã là cũ và không được người dùng quan tâm. Họ thường loại bỏ các tin nhắn cũ và chỉ chú ý đến các cập nhật gần nhất. Sự chuyển động của dữ liệu bây giờ hầu như là thực tế (real time) và tốc độ cập nhật thông tin đã giảm xuống đơn vị hàng mili giây. Vận tốc dữ liệu cao đại diện cho big data.[5]

Variety (Đa dạng)

Dữ liệu có thể được lưu trữ trong nhiều định dạng khác nhau. Ví dụ như: cơ sở dữ liệu, excel, csv, ms access hoặc thậm chí là tập tin văn bản (text). Đôi khi dữ liệu không ở dạng truyền thống như video, sms, pdf,… Thực tế dữ liệu thuộc nhiều định dạng và đó là thách thức của chúng ta. Sự đa dạng của dữ liệu đại diện cho big data.[5]

2.2.2 Cơ bản về kiến trúc của Big Data

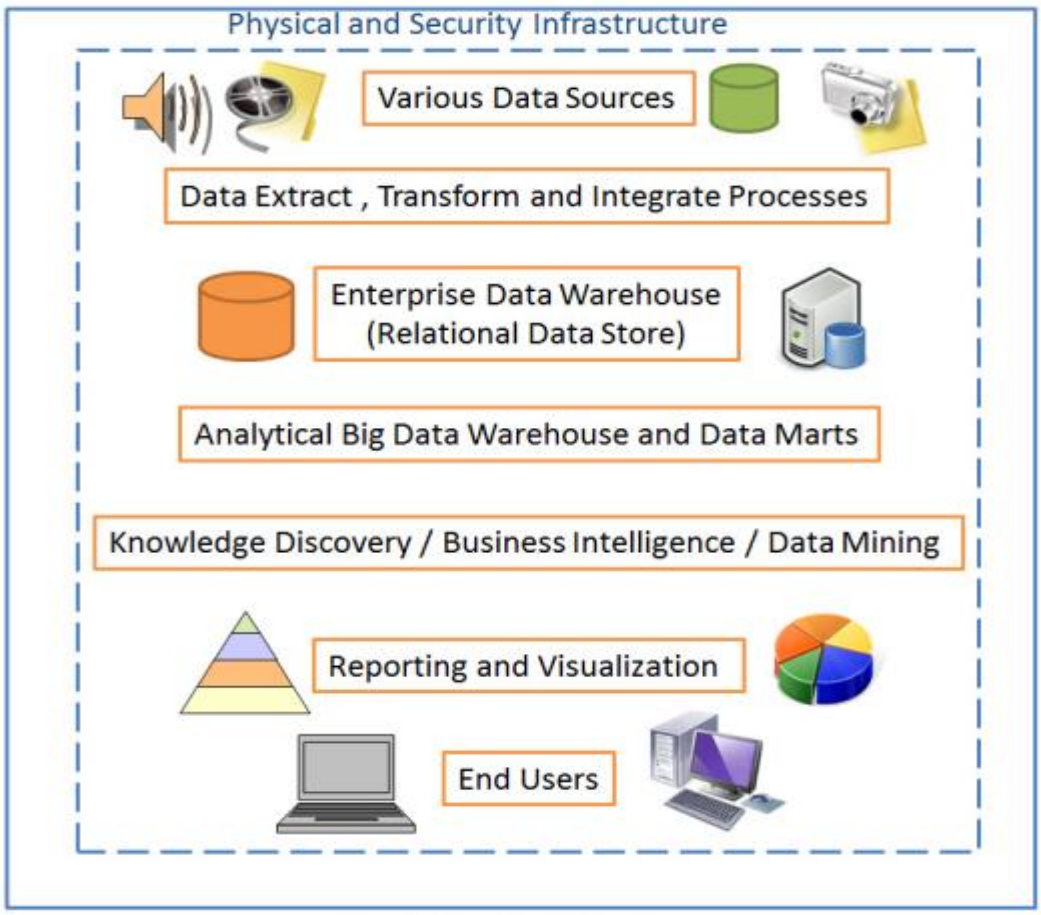
2.2.2.1 Chu kỳ của Big Data

Cũng giống các ứng dụng liên quan đến cơ sở dữ liệu khác, dự án Big Data cũng có chu kỳ phát triển của nó. Mô hình 3Vs đóng vai trò quan trọng trong việc quyết định kiến trúc của dự án Big Data. Dự án Big Data cũng có các đoạn (phase) như thu giữ dữ liệu, chuyển đổi, tích hợp, phân tích và xây dựng báo cáo . Các quá

trình này trông gần như giống nhau, nhưng do bản chất của dữ liệu, kiến trúc thường là hoàn toàn khác nhau.

2.2.2.2 Các thành phần của kiến trúc Big Data

Hoàn toàn không thể đưa ra giải pháp tối ưu nhất cho bất kỳ giải pháp big data nào trong 1 bài viết duy nhất, tuy nhiên, chúng ta có thể nói về các khối xây dựng cơ bản trong kiến trúc big data.



Hình 2.2.2.2 Kiến trúc Big Data [5]

Hình ảnh trên cho chúng ta cái nhìn tổng quan tốt về các thành phần khác nhau trong kiến trúc big data tương tác lẫn nhau. Trong big data, các nguồn dữ liệu khác nhau là 1 phần của kiến trúc do đó trích xuất, chuyển đổi và tích hợp (extract, transform and intergrate) là 1 trong những lớp quan trọng nhất của kiến trúc. Hầu hết các dữ liệu được lưu trữ trong quan hệ cũng như không quan hệ và các giải pháp kho dữ liệu. Theo nhu cầu kinh doanh, các dữ liệu đa dạng khác nhau được xử lý và

chuyển thành báo cáo trực quan với người dùng. Cũng giống như phần mềm, phần cứng cũng là phần quan trọng của kiến trúc big data. Trong kiến trúc big data, hạ tầng phần cứng vô cùng quan trọng và cần phải cài đặt ngăn chặn lỗi xảy ra, đảm bảo tính sẵn sàng cao.

NoSQL trong quản lý dữ liệu

NoSQL là 1 thuật ngữ rất nổi tiếng và nó thật sự có ý nghĩa là Not Relational SQL hay Not Only SQL. Điều này là do trong kiến trúc big data, dữ liệu ở định dạng bất kỳ. Để mang tất cả dữ liệu cùng nhau thì công nghệ mối quan hệ là không đủ, do các công cụ mới, kiến trúc và các thuật toán khác được phát minh sẽ nhận tất cả các loại dữ liệu. Những điều này được gọi chung là NoSQL.

2.3 Các ứng dụng của dữ liệu lớn

Có 4 lợi ích Big Data có thể mang lại: cắt giảm chi phí; giảm thời gian; tăng thời gian phát triển, tối ưu hóa sản phẩm; hỗ trợ con người đưa ra những quyết định đúng và hợp lý hơn. Thí dụ, khi mua sắm online trên eBay, Amazon hoặc những trang thương mại điện tử, các trang này sẽ đưa ra những sản phẩm gợi ý tiếp theo. Nếu chúng ta xem điện thoại, nó sẽ gợi ý mua thêm ốp lưng, pin dự phòng; hoặc khi mua áo thun sẽ có thêm gợi ý quần jean, dây nịt...

Do đó, nghiên cứu được sở thích, thói quen của khách hàng cũng gián tiếp giúp doanh nghiệp bán được nhiều hàng hóa hơn. Những thông tin về thói quen, sở thích này có được từ lượng dữ liệu khổng lồ các doanh nghiệp thu thập trong lúc khách hàng ghé thăm và tương tác với trang web của mình. Chỉ cần doanh nghiệp biết khai thác một cách có hiệu quả Big Data, nó không chỉ giúp tăng lợi nhuận cho chính họ mà giúp tiết kiệm thời gian cho khách hàng trong mua sắm.

Xu hướng Google rút ra từ những từ khóa tìm kiếm liên quan đến dịch H1N1 đã được chứng minh rất sát với kết quả do 2 hệ thống cảnh báo cúm độc lập Sentinel GP và HealthStat đưa ra. Dữ liệu của Flu Trends được cập nhật gần như theo thời gian thực, sau đó được đối chiếu với số liệu từ những trung tâm dịch bệnh ở nhiều nơi trên thế giới. Theo Oracle, việc phân tích Big Data và những dữ liệu dung lượng

lớn đã giúp các tổ chức kiếm được 10,66USD cho mỗi 1USD chi phí phân tích, tức gấp 10 lần.

Một trường học tại Hoa Kỳ có được sự tăng trưởng doanh thu 8 triệu USD mỗi năm, còn một công ty tài chính ẩn danh khác tăng 1.000% lợi nhuận trên tổng số tiền đầu tư của mình trong vòng 3 năm. Trong World Cup, Big Data cũng đưa ra dự báo đội tuyển Đức sẽ vô địch.

Thị trường Big Data được nhận định có giá trị tới 100 tỷ USD vào năm 2010 và đang không ngừng tăng với tốc độ chóng mặt. Chẳng hạn, hiện thế giới có tới 4,6 tỷ thuê bao điện thoại di động và có từ 1-2 tỷ người dùng internet. Từ năm 1990-2005, hơn 1 tỷ người trên thế giới tham gia vào tầng lớp trung lưu, tức nhu cầu lưu trữ và sử dụng thông tin của thế giới tăng lên nhiều lần.

Nếu để ý một chút, chúng ta sẽ thấy khi mua sắm online trên eBay, Amazon hoặc những trang tương tự, trang này cũng sẽ đưa ra những sản phẩm gợi ý tiếp theo cho bạn, ví dụ khi xem điện thoại, nó sẽ gợi ý cho bạn mua thêm ốp lưng, pin dự phòng; hoặc khi mua áo thun thì sẽ có thêm gợi ý quần jean, dây nịt... Do đó, nghiên cứu được sở thích, thói quen của khách hàng cũng gián tiếp giúp doanh nghiệp bán được nhiều hàng hóa hơn.

Vậy những thông tin về thói quen, sở thích này có được từ đâu? Chính là từ lượng dữ liệu khổng lồ mà các doanh nghiệp thu thập trong lúc khách hàng ghé thăm và tương tác với trang web của mình. Chỉ cần doanh nghiệp biết khai thác một cách có hiệu quả Big Data thì nó không chỉ giúp tăng lợi nhuận cho chính họ mà còn tăng trải nghiệm mua sắm của người dùng, chúng ta có thể tiết kiệm thời gian hơn nhờ những lời gợi ý so với việc phải tự mình tìm kiếm.

Người dùng cuối sẽ được hưởng lợi từ việc tối ưu hóa như thế, chứ bản thân người dùng khó mà tự mình phát triển hay mua các giải pháp để khai thác Big Data bởi giá thành của chúng quá đắt, có thể đến cả trăm nghìn đô. Ngoài ra, lượng dữ liệu mà chúng ta có được cũng khó có thể xem là "Big" nếu chỉ có vài Terabyte sinh ra trong một thời gian dài.

Ngoài ra, ứng dụng được Big Data có thể giúp các tổ chức, chính phủ dự đoán được tỉ lệ thất nghiệp, xu hướng nghề nghiệp của tương lai để đầu tư cho những hạng mục đó, hoặc cắt giảm chi tiêu, kích thích tăng trưởng kinh tế, v/v... thậm chí là ra phương án phòng ngừa trước một dịch bệnh nào đó, giống như trong phim World War Z, nước Israel đã biết trước có dịch zombie nên đã nhanh chóng xây tường thành ngăn cách với thế giới bên ngoài.

2.4 Các mô hình dữ liệu lớn

2.4.1 Hadoop Apache

2.4.1.1 Hadoop là gì?

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." [23]

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)."[22]

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.

[1]. Phần cứng thông thường: dịch từ thuật ngữ commodity hardware, tức các loại phần cứng thông thường, rẻ tiền. Các phần cứng này thường có khả năng hỏng hóc cao. Thuật ngữ này dùng để phân biệt với các loại phần cứng chuyên dụng đắt tiền, khả năng xảy ra lỗi thấp như các supermicrocomputer chẳng hạn.

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 .

2.4.1.2 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:

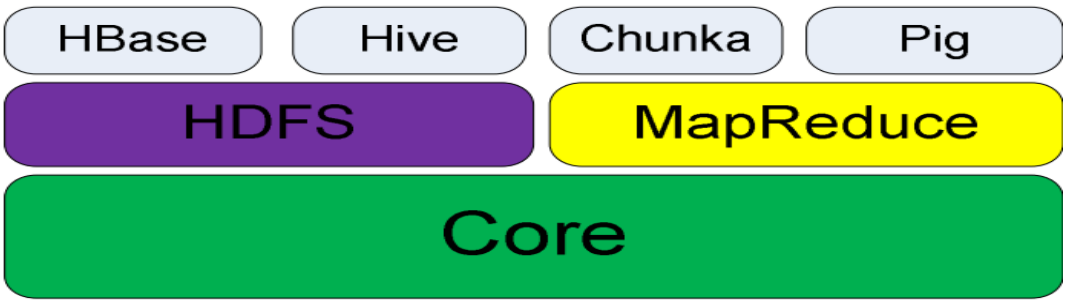
 Khả năng lưu trữ bị giới hạn: giả sử mỗi trang web cần 10kb đĩa cứng để lưu, thì với hơn 100 triệu trang ta cần 1 Tetabyte đĩa cứng, và với khối lượng hàng tỷ trang web đang có trên mạng thì cần có tới hàng chục petabye để lưu trữ.

 Tốc độ truy xuất chậm: với khối lượng dữ 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. [26]

2.4.1.3 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:



Hình 2.4.1.1 Cấu trúc các thành phần của Hadoop [22]

- Core: cung cấp các công cụ và giao diện cho hệ thống phân tán và các tiện ích I/O. Đây là phần lõi để xây dựng nên HDFS và MapReduce.
- MapReduce (MapReduce Engine): một framework giúp phát triển các ứng dụng phân tán theo mô hình MapReduce một cách dễ dàng và mạnh mẽ, ứng dụng phân tán MapReduce có thể chạy trên một cluster lớn với nhiều node.
- 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.

2.4.1.4 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. Hiện tại, facebook có 2 Hadoop cluster chính: một cluster 1100 máy với 8800 nhân và 12 Petabyte ổ cứng lưu trữ.

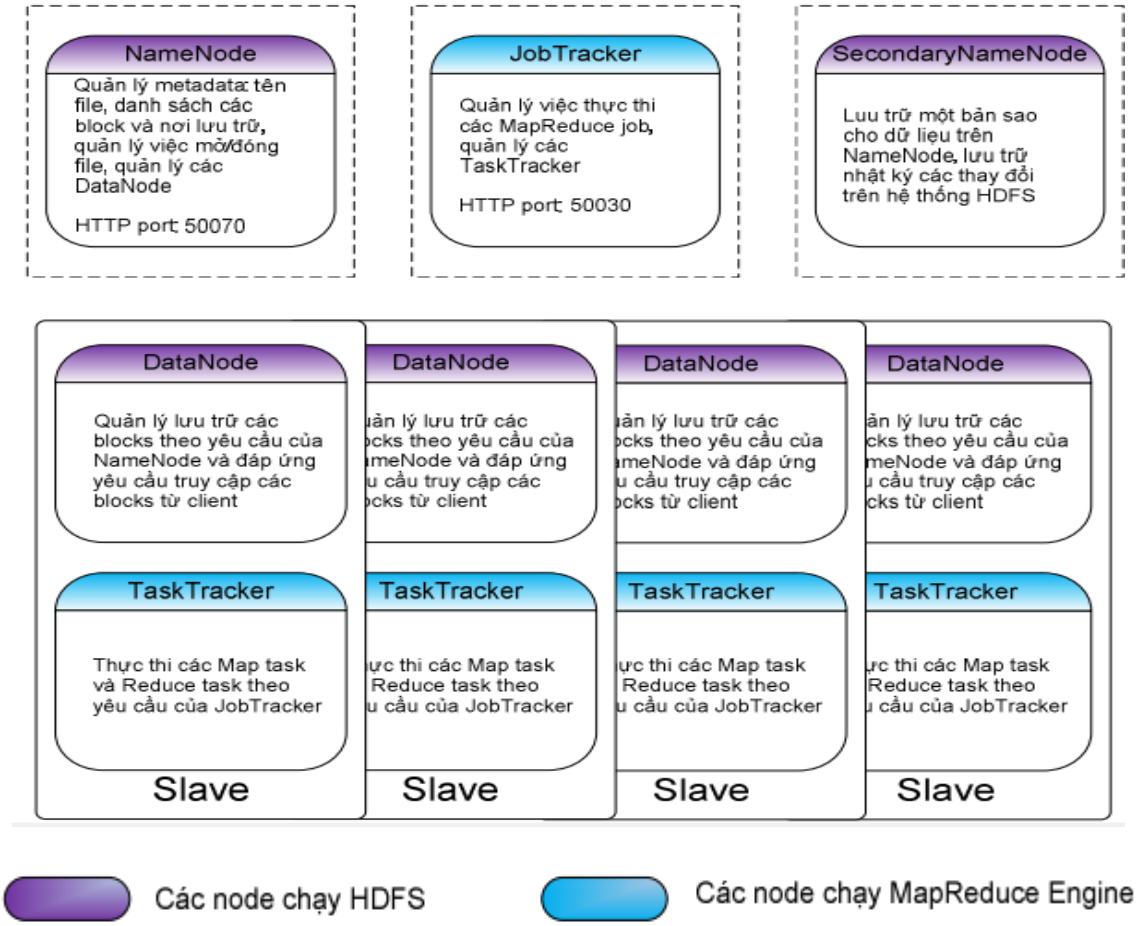
 A9.com-Amazon: Sử dụng Hadoop để đánh giá chỉ số tìm kiếm sản phẩm trên Amazon, xử lý đến hàng triệu Session mỗi ngày. Các cluster của A9.com có độ lớn từ 1-100 node.

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.

2.4.1.5 Tổng quan của một Hadoop cluster:

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:



Hình 2.4.1.2 Tổng quan một 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 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

. 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[4].

 TaskTracker: nhận các task từ JobTracker và thực hiện task. Ngoài ra trên một Hadoop cluster còn có SecondaryNameNode.

 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.

2.4.2 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 file phân tán phải quản lý được tình trạng hoạt động (live/dead) của các server tham gia vào hệ thống file.

3. Heartbeat: một loại thông điệp mà mỗi DataNode sẽ định kỳ gởi đến NameNode để xác nhận tình trạng hoạt động (death/live) của DataNode. Trên MapReduceEngine, các TaskTracker cũng dùng heartbeat để xác nhận tình trạng hoạt động của mình với JobTracker.

4. MapReduce Job: là một chương trình theo mô hình MapReduce được đệ trình lên để MapReduce Engine thực hiện. Xem phần MapReduce

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 sức mạnh của nó.

2.4.2.1 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 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.

2.4.2.2 Tổng quan thiết kế của HDFS

2.4.2.2.1 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.

 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.

2.4.2.2.2 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.

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:

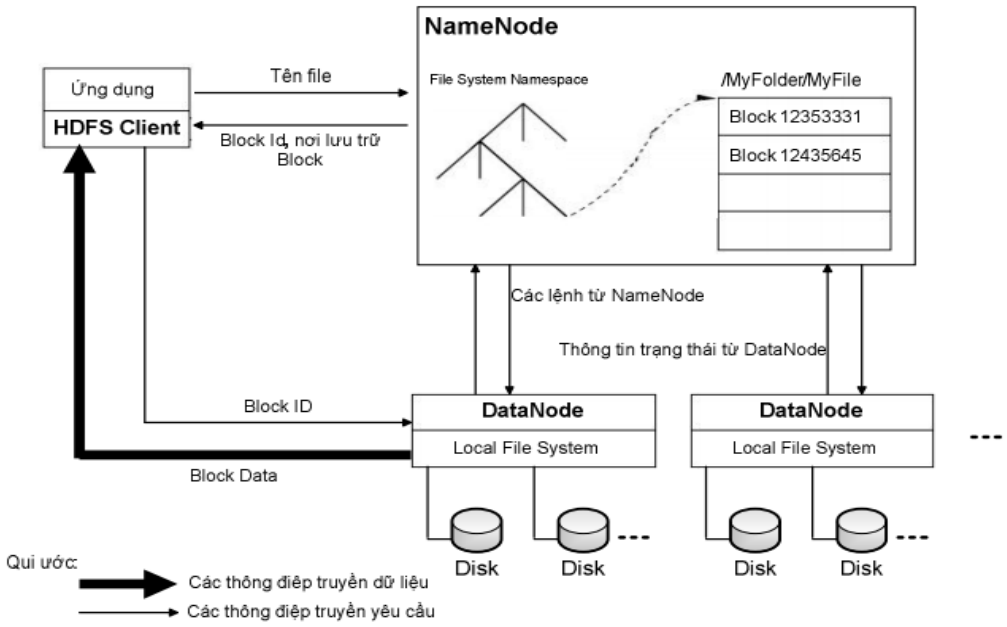
 File System Namespace: là hình ảnh cây thư mục của hệ thống file tại một thời điểm nào đó. File System namespace thể hiện tất các các file, thư mục có trên hệ thống file và quan hệ giữa chúng.

 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:



Hình 2.4.2.3 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 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 sau.

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.

2.4.2.2.3 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 Na meNode vào các quá trình đọc/ghi dữ liệu lên hệ thống để tránh tình trạng nút thắt cổ chai (bottle neck).

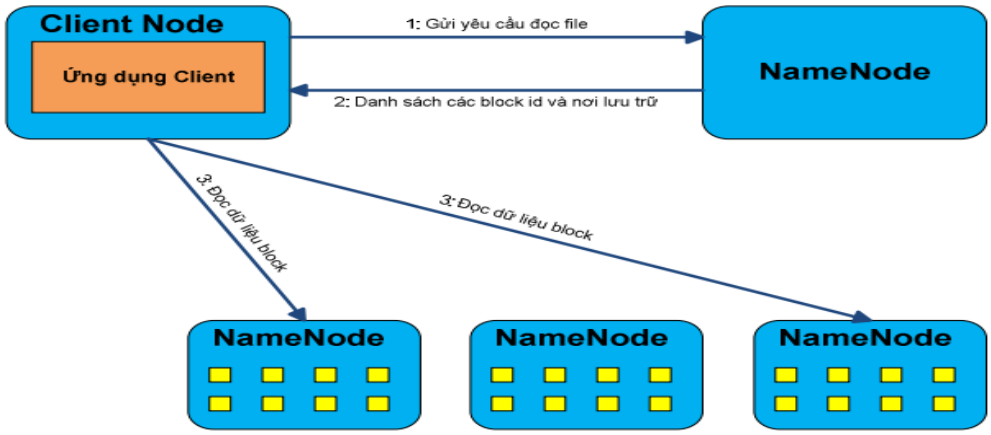
Client sẽ không bao giờ đọc hay ghi dữ liệu lên hệ thống thông qua NameNode. Thay vào đó, client sẽ hỏi NameNode xem nên liên lạc với DataNode

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.

2.4.2.2.3.1. Quá trình đọc file:

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



Hình 2.4.2.2.3.1 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).

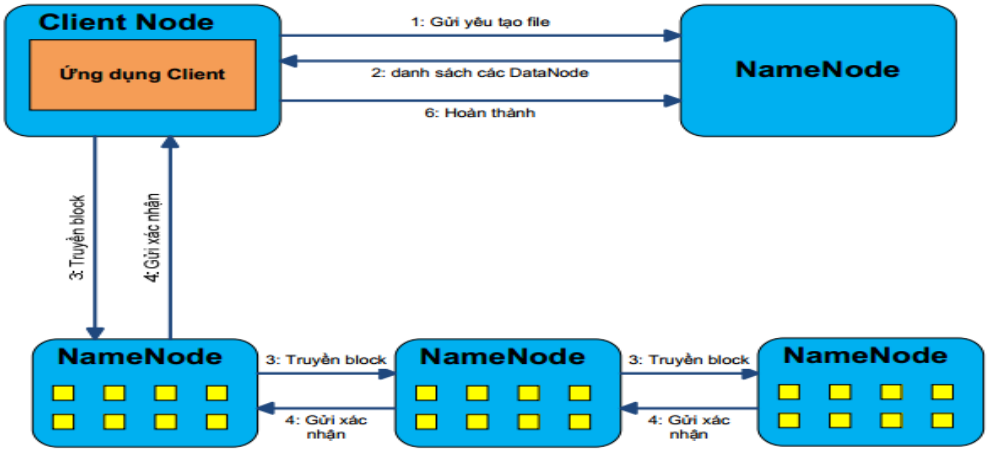
Lưu ý là với mỗi block ta có thể có nhiều DataNode lưu trữ các bản sao của block đó. Client sẽ chỉ đọc bản sao của block từ DataNode "gần" nhất. Việc tính khoảng cách giữa hai node trên cluster sẽ được trình bày ở phần 2.4.2.3.1 . Client sẽ thực hiện việc đọc các block lặp đi lặp lại cho đến khi block 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.

2.4.2.2.3.2 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.



Hình 2.4.2.2.3.2 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 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.

2.4.2.2.4 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 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 blockvà 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 với nhiều block nên sự bất tiện này là không đáng kể trong thực tiễn.

2.4.2.2.5 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.

2.4.2.2.5.1. 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.

2.4.2.2.5.2. 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.

2.4.2.2.5.3. 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 2.4.2.4.3 .

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.

2.4.2.3 Các tính năng của NameNode

2.4.2.3.1 Nhận biết cấu trúc topology của mạng

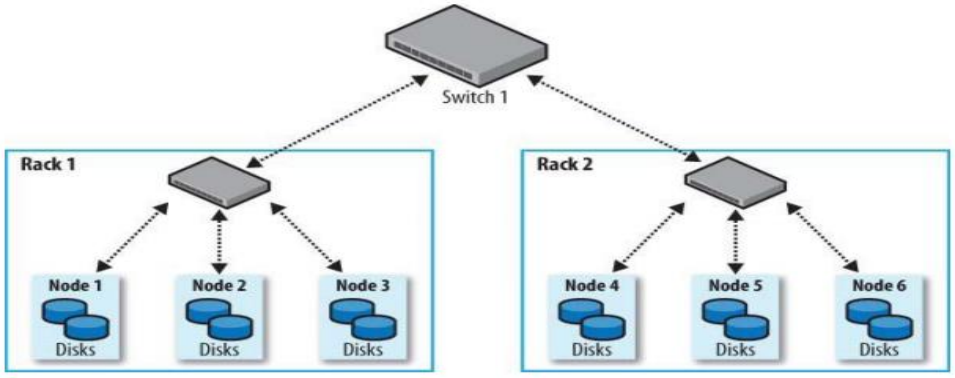
Trong bối cảnh xử lý dữ liệu với kích thước lớn qua môi trường mạng, việc nhận biết ra giới hạn về băng thông giữa các node là một yếu tố quan trọng để Hadoop đưa ra các quyết định trong việc phân bố dữ liệu và phân tán tính toán. Ý tưởng đo băng thông giữa hai node có vẻ như hợp lý, tuy nhiên làm được điều này là khá khó khăn(vì 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.



Hình 2.4.2.3.1 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.

Ta sẽ thấy rõ tác dụng của các giả định này trong các phần 2.4.2.3.2, 2.4.2.3.3.

2.4.2.3.2 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 2.4.2.2.3.2. , 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 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).

2.4.2.3.3 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 lậpvới 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 2.4.2.3.2 .

2.4.2.3.4 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 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.

2.4.2.4 Khả năng chịu lỗi và chẩn đoán lỗi của HDFS

2.4.2.4.1 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.

2.4.2.4.2 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.

2.4.2.4.3 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 cáchnào 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 điểmSecondaryNamenode 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.

2.4.2.4.4 Toàn vẹn dữ liệu trên HDFS

HDSF đảm bảo tính toàn vẹn của dữ liệu bằng cách thực hiện tạo checksum tất cả dữ liệu ghi lên nó và sẽ kiểm tra lại checksum mỗi khi đọc dữ liệu. 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. 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.

2.4.2.5 Các giao diện tương tác

2.4.2.5.1 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://<namenode>/<path> .

2.4.2.5.2 Giao diện java.

Hadoop được viết bằng Java. Vì vậy, tất cả 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.

2.4.2.5.3 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://<namenode>:50070/

2.4.2.6 Quản trị HDFS

2.4.2.6.1 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.

2.4.2.6.2 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 được tạo ra không có liên quan đến hạn ngạch vì nó không làm tăng dung lượng của thư mục cha.

2.4.3 Map reduce

2.4.3.1 Giới thiệu mô hình tính toán MapReduce

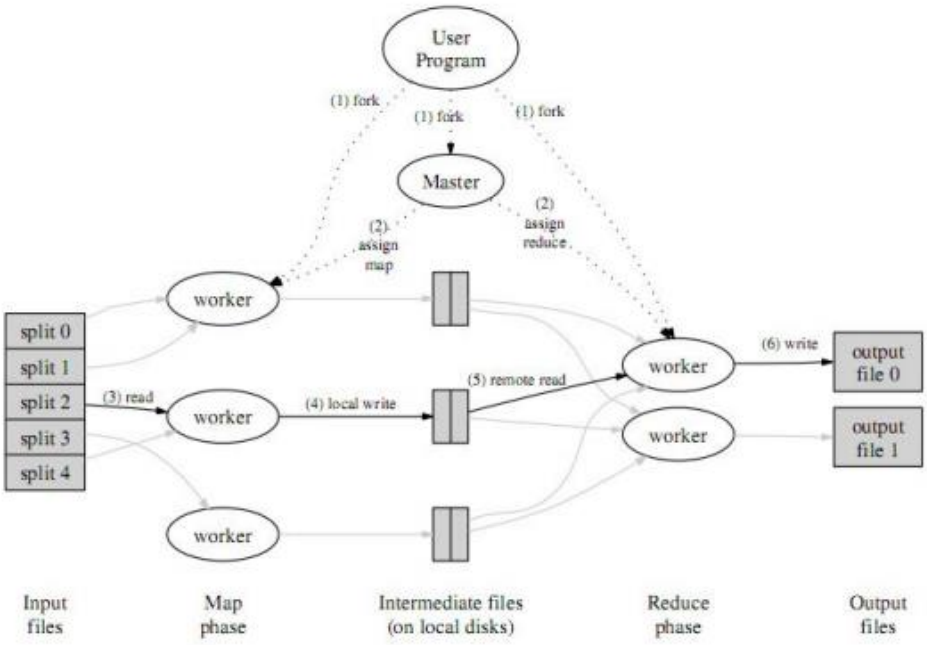
2.4.3.1.1 Nguyên nhân ra đời và lịch sử

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.

2.4.3.1.2 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.[21]



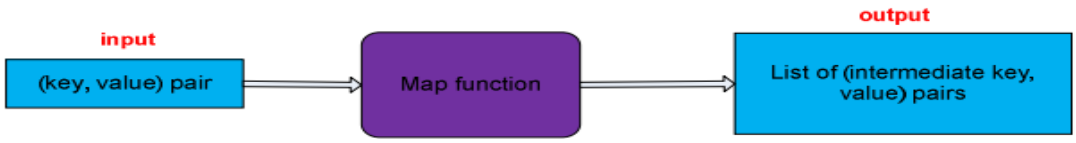
Hình 2.4.3.1 Mô hình Map Reduce của Google[5]

[5]. Nguồn: http://labs.google.com/papers/mapreduce.html

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 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 2.4.3.2.2.2.

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.

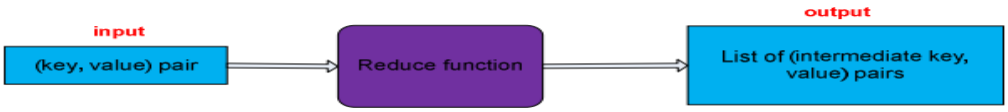
2.4.3.1.3 Hàm Map



Hình 2.4.3.1.3: 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).

2.4.3.1.4 Hàm Reduce



Hình 2.4.3.1.4: 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.

2.4.3.2 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.

2.4.3.2.1 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

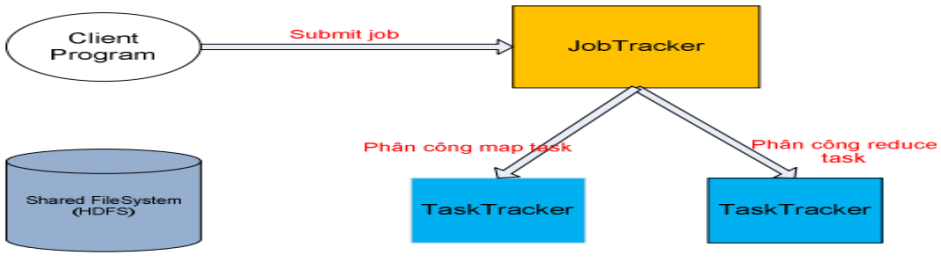
con nhỏ hơn, được định nghĩa là task. Và task này được gồm có 2 loại: map task và reduce task. Hiểu một cách đơn giản, map task là công việc con được thực thi thông qua việc sử dụng hàm map ,và tương tự với reduce task sẽ thực hiện hàm reduce. Điều này sẽ được làm rõ hơn ở mục "Cơ chế hoạt động của Hadoop 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

2.4.3.2.2 Kiến trúc MapReduce Engine

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



Hình 2.4.3.2.2.1: Kiến trúc các thành phần

Xét một cách trừu tượng, Hadoop MapReduce gồm 4 thành phần chính riêng biệt :

 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.

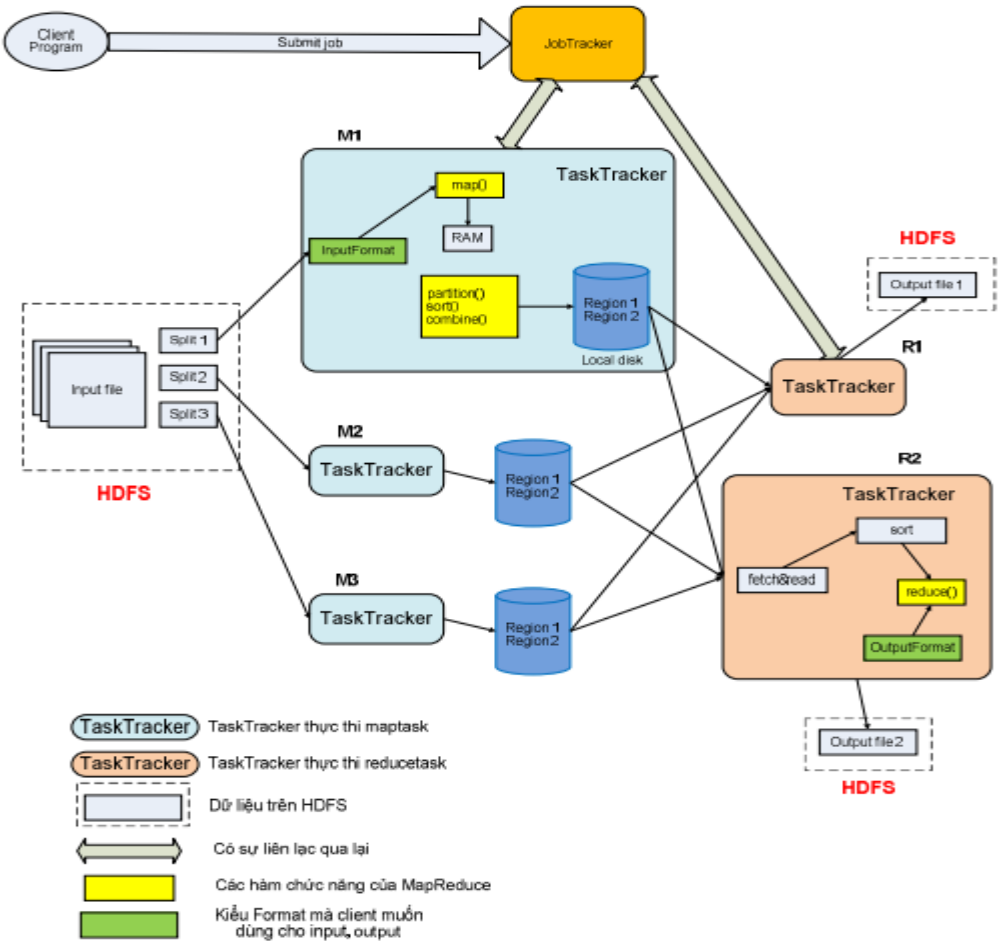
 TaskTracker: Đơn giản nó chỉ tiếp nhận maptask hay reducetask từ JobTracker để sau đó thực hiện. Và để giữ liên lạc với JobTracker, Hadoop Mapreduce cung cấp cơ chế gửi heartbeat từ TaskTracker đến JobTracker cho các nhu cầu như thông báo tiến độ của task do TaskTracker đó thực hiện, thông báo trạng thái hiện h ành 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.

2.4.3.2.2.2. Cơ chế hoạt động

Cơ chế hoạt động của Hadoop MapReduce mô tả cơchế 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.

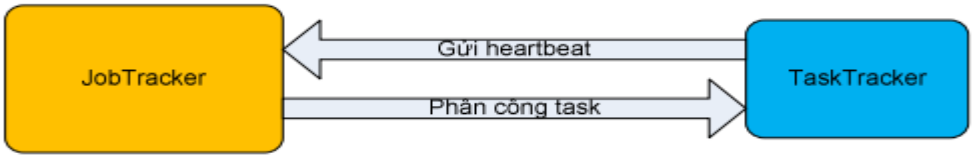
Đầu tiên chương trình client sẽ yêu cầu thực hiện job và kèm theo là dữ liệu input tới JobTracker. JobTracker sau khi tiếp nhận job này, nó sẽ thông báo ngược về chương trình client tình trạng tiếp nhận job. Khi chương trình client nhận được thông báo nếu tình trạng tiếp nhận hợp lệ thì nó sẽ tiến hành phân rã input này thành các split (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.



Hình 2.4.3.2.2.2: Cơ chế hoạt động của Hadoop MapReduce

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.

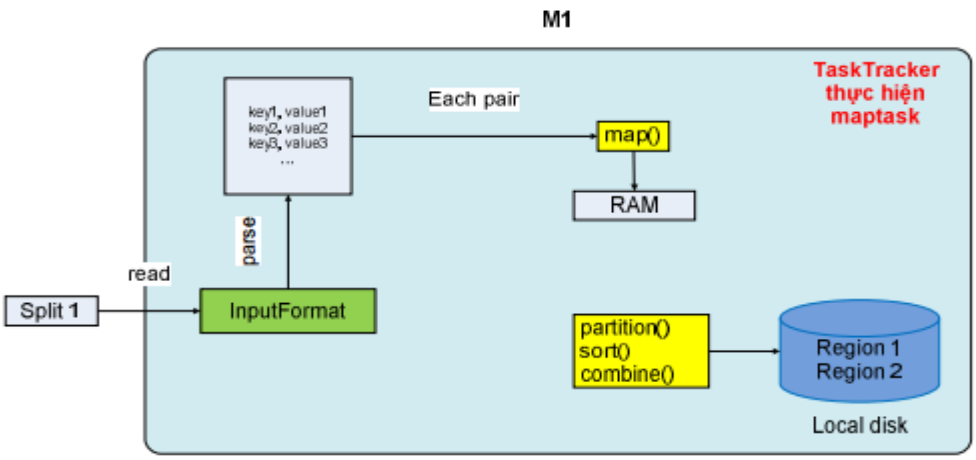
Với từng split thì nó sẽ tạo một maptask để thực thi, mặc nhiên số lượng maptask bằng với số lượng split. Còn đối với reduce task, số lượng reduce task được xác định bởi chương trình client. Bên cạnh đó, JobTracker còn lưu trữ thông tin trạng thái và tiến độ của tất cả các task.



Hình 2.4.3.2.2.3: Sự liên lạc đầu tiên giữa TaskTracker thực thi Maptask và JobTracker.

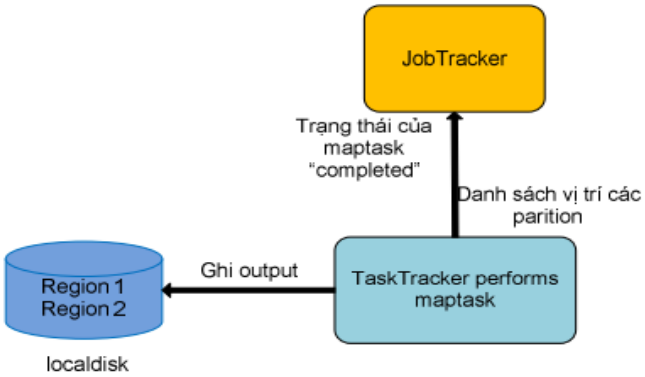
Ngay khi JobTracker khởi tạo các thông tin cần thiết để chạy job, thì bên cạnh đó các TaskTracker trong hệ thống sẽ gửi các heartbeat đến JobTracker. Hadoop cung cấp cho các TaskTracker cơ chế gửi heartbeat đến JobTracker theo chu kỳ thời gian nào đó, thông tin bên trong heartbeat này cho phép JobTrack biết được TaskTracker này có thể thực thi task hay không. 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 heap bên trong TaskTracker này.

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



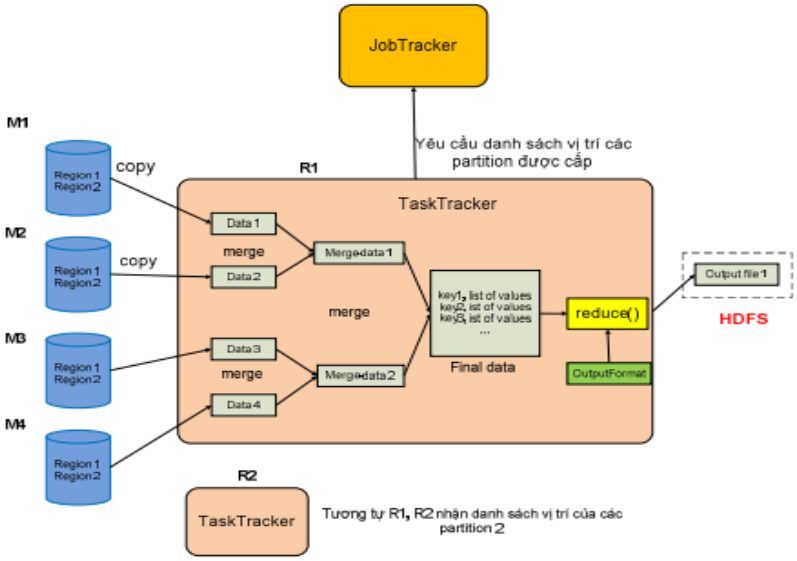
Hình 2.4.3.2.2.4: Cơ chế hoạt động của Map task

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 key 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 2.4.3.2.2.5: TaskTracker hoàn thành Map task

Trước khi ghi xuống local disk, các dữ liệu output này sẽ được phân chia vào các partition (region) dựa vào hàm partition, 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 combine 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. Đó là toàn bộ quá trình TaskTracker thực hiện một maptask



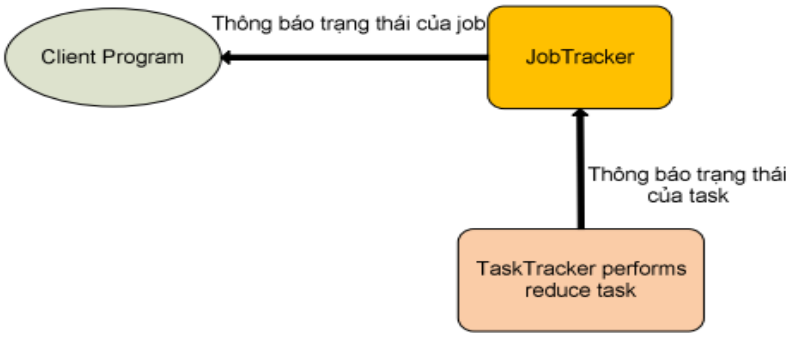
Hình 2.4.3.2.2.6: Cơ chế hoạt động của Reduce task

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 .

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 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 nạp (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 merge 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.



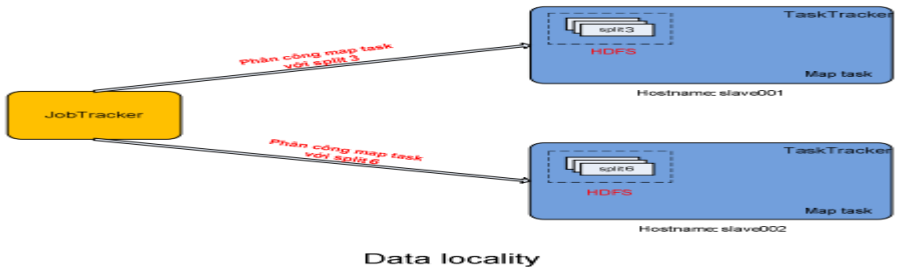
Hình 2.4.3.2.2.7: TaskTracker hoàn thành Reduce task

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 2.4.3.2.2.5: 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).

*2.4.3.2.2.3. MapReduce và HDFS (Các đặc điểm tối ưu của MapReduce khi kết hợp với HDFS):*HDFS chỉ là hệ thống file phân tán với các cơ chế quản lý bên trong nó. Lý do 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).

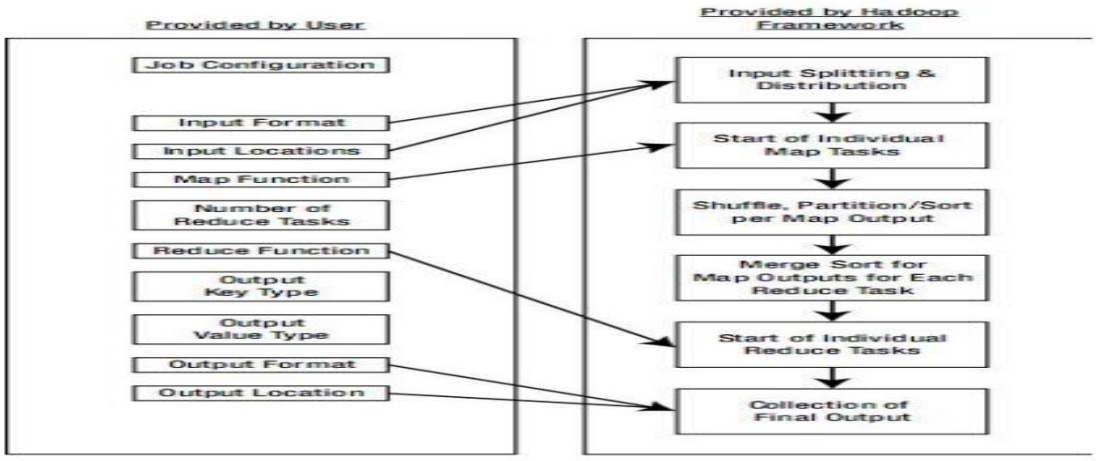


Hình 2.4.3.2.2.8: Data locality

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 Hadoop định nghĩa là 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.

2.4.3.2.3 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).



Hình 2.4.3.2.3: 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.

o Sau đó người dùng phải truyền vào đường dẫn của dữ liệu input.

o 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

2.4.3.2.4 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.

Chương 3: Mô hình điều khiển truy xuất dữ liệu.

3.1 Tổng quan điều khiển truy cập

3.1.1 Giới thiệu điều khiển truy cập

3.1.1.1 Khái niệm về truy cập và điều khiển truy cập

Truy cập (access) là khả năng tương tác giữa chủ thể(subject) và đối tượng (object).

Điều khiển truy cập là quá trình mà trong đó người dùng được nhận dạng và trao quyền truy cập đến các thông tin, các hệ thống và tài nguyên. Điều khiển truy cập tạo nên khả năng cho chúng ta có thể cấp phép hoặc từ chối một chủ thể - một thực thể chủ động, chẳng hạn như một người hay một quy trình nào đó - sử dụng một đối tượng - một thực thể thụ động, chẳng hạn như một hệ thống, một tập tin - nào đó trong hệ thống.

Có ba khái niệm cơ bản trong mọi ngữ cảnh điều khiển truy cập, bao gồm:

- Chính sách (policy): Là các luật do bộ phận quản trị tài nguyên đề ra.
- Chủ thể (subject): Có thể là người sử dụng, mạng, các tiến trình hay các ứng dụng yêu cầu được truy cập vào tài nguyên.
- Đối tượng (object): Là các tài nguyên mà chủ thể được phép truy cập.

3.1.1.2. Các thành phần cơ bản của điều khiển truy cập

Các hệ thống điều khiển truy cập (Access control systems)

Một hệ thống điều khiển truy cập hoàn chỉnh bao gồm 3 thành phần:

- Các chính sách (Policies): Các luật được đưa ra bởi bộ phận quản lý tài nguyên quy định phương thức truy cập vào tài nguyên.
- Các thủ tục (Procedures): Các biện pháp phi kỹ thuật được sử dụng để thực thi các chính sách.
- Các công cụ(Tools): Các biện pháp kỹ thuật được sử dụng để thực thi các chính sách.

Các chủ thể điều khiển truy cập (Access control subjects)

Các chủ thể (subject) trong ngữ cảnh điều khiển truy cập là một cá nhân hoặc một ứng dụng đang yêu cầu truy xuất vào một tài nguyên như mạng, hệ thống file hoặc máy in. Có 3 loại chủ thể:

- Đã xác thực : Là những người có sự ủy quyền hợp pháp được phép truy cập vào tài nguyên.
- Chưa xác thực: Là những người chưa có sự ủy quyền hợp pháp hoặc không có quyền để truy cập vào tài nguyên.
- Chưa biết (Unknown): Những người chưa rõ, không xác định về quyền hạn truy cập.

Các đối tượng điều khiển truy cập (Access control objects)

Ba danh mục chính của đối tượng cần được bảo vệ bằng điều khiển truy cập:

- Thông tin: Là tất cả dữ liệu.
- Công nghệ: Là các ứng dụng, hệ thống và mạng.
- Địa điểm vật lý: Như các tòa nhà, văn phòng...

Thông tin là dữ liệu phổ biến nhất trong các chính sách điều khiển truy cập của công nghệ thông tin. Có thể đặt mật khẩu cho các ứng dụng và cơ sở dữ liệu để hạn chế việc truy cập. Các đối tượng công nghệ cũng quan trọng bởi vì khi có thể truy nhập vào các đối tượng công nghệ thì cũng có khả năng truy nhập vào các thông tin.

3.1.1.3 Tiến trình điều khiển truy cập

Ba bước để thực hiện điều khiển truy cập:

- Nhận dạng (Identification): Xử lý nhận dạng một chủ thể khi truy cập vào hệ thống.
- Xác thực (Authentication): Chứng thực nhận dạng chủ thể đó.
- Trao quyền (Authorization): Gán quyền được phép hoặc không được phép truy cập vào đối tượng.

Nhận dạng (Identification)

Nhận dạng là phương pháp người dùng báo cho hệthống biết họlà ai (chẳng hạn như bằng cách sử dụng tên người dùng). Bộ phận nhận dạng người dùng của một hệ thống điều khiển truy cập thường là một cơ chế tương đối đơn giản.

Xác thực (Authentication)

Xác thực là một quy trình xác minh "nhận dạng" của một người dùng - chẳng hạn bằng cách so sánh mật khẩu mà người dùng đăng nhập với mật khẩu được lưu trữ trong hệ thống đối với một tên người dùng cho trước nào đó. Có nhiều phương pháp xác thực một chủ thể. Một số phương pháp xác thực được sử dụng phổ biến:

- Mật khẩu.
- Token.
- Khóa chia sẻ bí mật (shared secret).

Trao quyền (Authorization)

Khi một subject đã được "nhận dạng" và "xác thực" được phép truy cập vào hệ thống, hệ thống điều khiển truy cập phải xác định subject này được cấp quyền hạn gì khi truy cập vào tài nguyên được yêu cầu. Trao quyền cấp các quyền phù hợp theo định nghĩa từ trước của hệ thống cho subject truy nhập vào object.

3.1.2. Các kiểu xác thực

Có ba kiểu xác thực các chủ thể được sử dụng phổ biến nhất:

- Xác thực dựa trên cái người sử dụng biết (something you know).
- Xác thực dựa trên những thứ người sử dụng có (something you have).
- Xác thực dựa trên những thứ người sử dụng sở hữu bẩm sinh (something you are).
- Xác thực dựa trên cái người sử dụng biết như mật khẩu (password), mật ngữ(pass phrase) hoặc mã số định danh cá nhân (PIN)…

3.1.3. Các nguy cơ và các điểm yếu của điều khiển truy cập

3.1.3.1. Các nguy cơ(threats)

Có ba nguy cơchính đối với bất kỳ hệ thống điều khiển truy cập:

- Phá mật khẩu (password cracking).
- Chiếm quyền điều khiển (heightened access).
- Social engineering.

Phá mật khẩu (Password Cracking)

Người quản trị hệ thống sẽ thiết lập các luật cho mật khẩu để đảm bảo người sử dụng tạo mật khẩu an toàn nhất. Kẻ tấn công có thể sử dụng kết hợp kỹ thuật tấn công Brute force và các thuật toán tinh vi để phá mật khẩu, truy cập vào hệ thống một

cách bất hợp pháp. Các chính sách để đảm bảo mật khẩu đạt độ khó tránh bị phá bao gồm: mật khẩu phải tối thiểu 8 ký tự bao gồm chữ cái hoa, chữ cái thường, số và ký tự đặc biệt. Bên cạnh đó ý thức người sử dụng cũng cần được nâng cao như định kỳ nên thay đổi mật khẩu, đặt mật khẩu phải đạt độ khó nhưng dễ nhớ...

Chiếm quyền điều khiển (Heightened Access)

Kẻ tấn công có thể khai thác các điểm yếu trên hệ điều hành, dùng công cụ để phá mật khẩu của người dùng và đăng nhập vào hệ thống trái phép, sau đó sẽ tiếp theo tìm cách nâng quyền truy cập ở mức cao hơn. Cơ hội là các thông tin có giá trị trên hệ thống (như các dữ liệu nhạy cảm) đã được bảo vệ bởi việc phân quyền cho nhóm và file không cho phép mọi người sử dụng có thể đọc và viết chúng.

Social Engineering

"Social engineering" sử dụng sự ảnh hưởng và sự thuyết phục để đánh lừa người dùng nhằm khai thác các thông tin có lợi cho cuộc tấn công hoặc thuyết phục nạn nhân thực hiện một hành động nào đó. Social engineer (người thực hiện công việc tấn công bằng phương pháp social engineering) thường sử dụng điện thoại hoặc internet để dụ dỗ người dùng tiết lộ thông tin nhạy cảm. Bằng phương pháp này, Social engineer tiến hành khai thác các thói quen tự nhiên của người dùng, hơn là tìm các lỗ hổng bảo mật của hệ thống.

3.1.3.2. Các điểm yếu

Hầu hết các hệ thống bảo mật đều có các điểm yếu nào đó và các hệ thống điều khiển truy cập cũng không phải ngoại lệ. Điểm yếu chính khi sử dụng mật khẩu trong điều khiển truy cập chính là việc sử dụng mật khẩu "yếu", dễ đoán, dễ phá. Để hạn chế điểm yếu này thì việc sử dụng mật khẩu cần tuân theo các chính sách như việc tạo mật khẩu phải ít nhất 8 ký tự trở lên trong đó có chữ hoa, chữ thường, số, dễ nhớ và khó đoán. Việc sử dụng các thiết bị phần cứng kết hợp với sử dụng mật khẩu như security token và thiết bị sinh mật khẩu một lần (OTP) cũng là các giải pháp hạn chế điểm yếu về mật khẩu của điều khiển truy cập.

3.1.3.3. Đánh giá ảnh hưởng của các nguy cơ và điểm yếu đối với điều khiển truy cập.

Trên cơ sở phân tích các nguy cơ và điểm yếu của điều khiển truy cập, chúng ta có thể đánh giá các tác động của chúng. Có hai cách đánh giá: Đánh giá theo định lượng và theo định tính.

Đánh giá theo định lượng là việc ước lượng các chi phí phải trả để khắc phục hậu quả của các tấn công, phá hoại và khôi phục dữ liệu.

Đánh giá theo định tính: Đánh giá rủi ro về chất lượng đưa vào tài khoản phi tài chính rủi ro đối với một tổ chức.

3.1.4. Một số ứng tiêu biểu của điều khiển truy cập

3.1.4.1. Kerberos

Kerberos là hệ xác thực dựa trên nguyên lý mã hóa sử dụng khóa mật. Trong hệ Kerberos, một bên thứ ba được tin cậy cấp khóa phiên để bên người dùng và bên cung cấp dịch vụ có thể trao đổi thông tin với nhau trên mạng một cách an toàn. Đây là một công nghệ đã chín muồi và được sử dụng rộng rãi, tuy còn một số mặt hạn chế đang được tiếp tục khắc phục.

3.1.4.2. Đăng nhập một lần

Đăng nhập một lần (Single Sign On hay SSO) là giải pháp sử dụng một dịch vụ chứng thực trung tâm để chứng thực người dùng cho rất nhiều dịch vụ khác. Vì vậy, chỉ cần một tài khoản, khách hàng có thể đăng nhập và sử dụng rất nhiều dịch vụ chạy trên các máy chủ và tên miền khác nhau. Giải pháp này có thể giúp doanh nghiệp giảm thiểu chi phí, tăng cường an ninh và đặc biệt là mang lại sự thuận tiện cho khách hàng.

3.1.4.3. Tường lửa

Thuật ngữ Firewall có nguồn gốc từ một kỹ thuật thiết kế trong xây dựng để ngăn chặn, hạn chế hoả hoạn. Trong công nghệ mạng thông tin, Firewall là một kỹ thuật được tích hợp vào hệ thống mạng để chống sự truy cập trái phép nhằm bảo vệ các tài nguyên mạng nội bộ cũng như hạn chế sự xâm nhập của một số thông tin không mong muốn. Cũng có thể hiểu rằng Firewall là một cơ chế để bảo vệ mạng tin tưởng (trusted network) khỏi các mạng không tin tưởng (untrusted network).

3.2 Các điều khiển truy cập thông dụng

3.2.1. Điều khiển truy cập tùy quyền (DAC - Discretionary Access Control)

DAC hay còn gọi là mô hình điều khiển truy cập tùy quyền là một phương pháp nhằm hạn chế truy cập các đối tượng trên cơ sở nhận dạng và nhu cầu cần biết của nhiều người dùng và/hoặc của một nhóm các đối tượng trực thuộc. Phương pháp điều khiển truy cập được coi là tùy quyền là vì một chủ thể với một quyền truy cập nào đó có thể chuyển nhượng quyền truy cập (trực tiếp hay gián tiếp) sang bất cứ một chủ thể nào khác trong hệ thống. Nói cách khác, kỹ thuật này cho phép người dùng có toàn quyền quyết định quyền truy cập được công nhận cho các tài nguyên của họ, có nghĩa là họ có thể (tình cờ hay cố ý) cấp quyền truy cập cho những người dùng bất hợp pháp.

Hiện nay, các hệ điều hành thường hỗ trợ năm cơ chế cơ bản:

- Các khả năng (Capabilities).
- Hồ sơ (Profiles).
- Danh sách điều khiển truy cập (Access Control Lists – ACLs).
- Các bit bảo vệ(Protection bits).
- Mật khẩu (Passwords).

Các khả năng (Capabilites) : Các khả năng tương ứng với các hàng của ma trận điều khiển truy cập. Khi phương pháp này được sử dụng, liên kết với mỗi tiến trình là một danh sách các đối tượng có thể được truy cập, cùng với một dấu hiệu của hoạt động nào được cho phép, nói cách khác, là miền của nó. Danh sách này được gọi là một danh sách các khảnăng hoặc Clist và các thành phần trên đó được gọi là những khả năng.

Các hồ sơ(Profiles): Profiles được triển khai trên nhiều dạng hệ thống, sử dụng một danh sách bảo vệ đối tượng kết hợp với từng người dùng. Nếu một người dùng được truy cập đến nhiều đối tượng được bảo vệ, profiles có thể rất lớn và khó quản lý. Viêc tạo, xóa và thay đổi truy cập đến đối tượng được bảo vệ yêu cầu nhiều thao tác khi nhiều profile của người dùng phải được cập nhật. Việc xóa một đối tượng có thể yêu cầu một vài thao tác xác định một người dùng có các đối tượng ở trong profile của

mình. Với profile, để trả lời câu hỏi "ai có quyền truy cập vào đối tượng được bảo vệ" là rất khó. Nhìn chung, không nên triển khai profiles trong hệ thống DAC.

Access control lists (ACLs) : Danh sách điều khiển truy cập (Access control lists – ACLs) là danh sách mô tả việc liên kết các quyền truy nhập của người dùng với mỗi đối tượng. Đó là một danh sách có chứa tất cả các miền có thể truy cập vào các đối tượng. Thông thường trong các tài liệu bảo mật, người dùng được gọi là các chủ thể (subject), để tương ứng với những thứ họ sở hữu, các đối tượng, chẳng hạn như các file. Mỗi tập tin có một bản ghi ACL liên kết với nó. ACL không thay đổi nếu người dùng khởi động một tiến trình hoặc 100 tiến trình. Quyền truy nhập được gán cho chủ sở hữu, không phải gán trực tiếp cho tiến trình.

Các bit bảo vệ(Protection bits) : Các bit bảo vệ đặc trưng ma trận điều khiển truy cập theo cột. Trong cơ chế "bit bảo vệ" trên các hệ thống như UNIX, bit bảo vệ cho mỗi đối tượng được sử dụng thay vì liệt kê danh sách người dùng có thể truy cập vào đối tượng. Trong UNIX các bit bảo vệ chỉ ra hoặc mọi người, nhóm đối tượng hoặc người sở hữu mới có các quyền để truy cập đến đối tượng được bảo vệ. Người tạo ra đối tượng được gọi là chủ sở hữu (owner), chủ sở hữu này có thể thay đổi bit bảo vệ. Hệ thống không thể cho phép hay không cho phép truy cập tới một đối tượng được bảo vệ trên bất kỳ người dùng nào.

Mật khẩu : Mật khẩu bảo vệ các đối tượng đại diện cho ma trận kiểm soát truy cập của hàng. Nếu mỗi người sử dụng sở hữu mật khẩu của mình cho từng đối tượng, sau đó mật khẩu là một vé cho đối tượng, tương tự như một hệ thống khả năng. Trong hầu hết các cài đặt thực hiện bảo vệ mật khẩu, chỉ có một mật khẩu cho mỗi đối tượng hoặc mật khẩu mỗi đối tượng cho mỗi chế độ truy cập tồn tại.

3.2.2 Điều khiển truy cập bắt buộc (MAC – Mandatory access control)

Kiểm soát truy nhập bắt buộc (Mandatory Access Control - MAC) là một chính sách truy nhập không do cá nhân sở hữu tài nguyên quyết định, mà do hệ thống quyết định. MAC được dùng trong các hệ thống đa cấp, là những hệ thống xử lý các loại dữ liệu nhạy cảm, như các thông tin được phân loại theo mức độ bảo mật trong cơ quan chính phủ và trong quân đội. Một hệ thống đa cấp là một hệ thống

máy tính duy nhất chịu trách nhiệm xử lý nhiều cấp thông tin nhạy cảm giữa các chủ thể và các đối tượng trong hệ thống.

Khái niệm MAC được hình thức hoá lần đầu tiên bởi mô hình Bell và LaPadula. Mô hình này hỗ trợ MAC bằng việc xác định rõ các quyền truy nhập từ các mức nhạy cảm kết hợp với các chủ thể và đối tượng. Mô hình toàn vẹn Biba được đưa ra năm 1977 tại tổng công ty MITRE. Một năm sau khi mô hình Bell-LaPadula được đưa ra. Các động lực chính cho việc tạo mô hình này là sự bất lực của mô hình Bell-LaPadula để đối phó với tính toàn vẹn của dữ liệu.

Mô hình Bell-LaPadula :Mô hình Bell-La Padula là mô hình bảo mật đa cấp được sử dụng rộng rãi nhất. Mô hình này được thiết kế để xử lý an ninh quân sự, nhưng nó cũng có thể áp dụng cho các tổ chức khác. Một tiến trình chạy nhân danh một người sử dụng có được mức độ bảo mật của người dùng đó. Vì có nhiều mức độ bảo mật, mô hình này được gọi là một hệ thống đa bảo mật.

Mô hình Bell-La Padula có những quy định về thông tin có thể lưu thông:

- Tài nguyên bảo mật đơn giản: Một tiến trình đang chạy ở mức độ bảo mật k có thể đọc các đối tượng chỉ ở cùng mức hoặc thấp hơn.
- Tài nguyên *: Một tiến trình đang chạy ở mức độ bảo mật k chỉ có thể ghi các đối tượng ở cùng cấp độ hoặc cao hơn.

Mô hình Biba : Mô hình toàn vẹn Biba được đưa ra năm 1977 tại tổng công ty MITRE. Một năm sau khi mô hình Bell-LaPadula được đưa ra. Các động lực chính cho việc tạo mô hình này là sự bất lực của mô hình Bell-LaPadula để đối phó với tính toàn vẹn của dữ liệu. Mô hình này chú trọng vào tính toàn vẹn, dựa trên 2 quy tắc:

- Đối tượng không được xem các nội dung ở mức an ninh toàn vẹn thấp hơn (no readdown).
- Đối tượng không được tạo/ghi các nội dung ở mức an ninh toàn vẹn cao hơn (no write-up). Vấn đề với mô hình Padula Bell-La là nó đã được đưa ra để giữ bí mật, không đảm bảo tính toàn vẹn của dữ liệu. Để đảm bảo tính toàn vẹn của dữ liệu, các nguyên tắc sau được áp dụng:

- Nguyên tắc toàn vẹn đơn giản: Một tiến trình đang chạy ở mức độ bảo mật k có thể chỉ có thể ghi lên các đối tượng ở cùng mức hoặc thấp hơn (không viết lên mức cao hơn).
- Tính toàn vẹn tài nguyên: Một tiến trình đang chạy ở mức độ bảo mật k chỉ thể đọc các đối tượng ở cùng mức hoặc cao hơn (không đọc xuống mức thấp hơn).

3.2.3 Mô hình điều khiển truy cập trên cơ sở vai trò (RBAC-Role-based Access Control)

Khái niệm điều khiển truy cập dựa trên vai trò (Role-Based Access Control) bắt đầu với hệ thống đa người sử dụng và đa ứng dụng trực tuyến được đưa ra lần đầu vào những năm 70. Ý tưởng trọng tâm của RBAC là permission (quyền hạn) được kết hợp với role (vai trò) và user (người sử dụng) được phân chia dựa theo các role thích hợp. Điều này làm đơn giản phần lớn việc quản lý những permission. Tạo ra các role cho các chức năng công việc khác nhau trong một tổ chức và user cũng được phân các role dựa vào trách nhiệm và trình độ của họ. Những role được cấp các permission mới vì các ứng dụng gắn kết chặt chẽ với các hệ thống và các permission được hủy khỏi các role khi cần thiết.

Nền tảng và động lực

Với RBAC, người ta có thể xác định được các mối quan hệ role - permission. Điều này giúp cho việc gán cho các user tới các role xác định dễ dàng. Các permission được phân cho các role có xu hướng thay đổi tương đối chậm so với sự thay đổi thành viên những user các role.

Chính sách điều khiển truy cập được thể hiện ở các thành tố khác nhau của RBAC như mối quan hệ role - permission, mối quan hệ user – role và mối quan hệ role – role. Những thành tố này cùng xác định xem liệu một user cụ thể có được phép truy cập vào một mảng dữ liệu trong hệ thống hay không. RBAC không phải là giải pháp cho mọi vấn đề kiểm soát truy cập. Người ta cần những dạng kiểm soát truy cập phức tạp hơn khi xử lí các tình huống mà trong đó chuỗi các thao tác cần được kiểm soát.

Các mô hình tham chiếu

Để hiểu các chiều khác nhau của RBAC, cần xác định 4 mô hình RBAC khái niệm. RBAC0, mô hình cơ bản nằm ở dưới cùng cho thấy đó là yêu cầu tối thiểu cho bất kì một hệ thống nào hỗ trợ RBAC. RBAC1 và RBAC2 đều bao gồm RBAC0 nhưng có thêm một số nét khác với RBAC0. Chúng được gọi là các mô hình tiên tiến. RBAC1 có thêm khái niệm cấp bậc role (khi các role có thể kế thừa permission từ role khác). RBAC2 có thêm những ràng buộc (đặt ra các hạn chếchấp nhận các dạng của các thành tố khác nhau của RBAC). RBAC1 và RBAC2 không so sánh được với nhau. Mô hình hợp nhất RBAC3 bao gồm cả RBAC1 và RBAC2 và cả RBAC0 nữa.
Mô hình cơ sở: Mô hình cơ sở RBAC0 không phải là một trong 3 mô hình tiên tiến. Mô hình có 3 nhóm thực thể được gọi là User (U), Role (R), Permission (P) và một tập hợp các Session (S) .

User trong mô hình này là con người. Khái niệm user sẽ được khái quát hóa bao gồm cả các tác nhân thông minh và tự chủ khác như robot, máy tính cố định, thậm chí là các mạng lưới máy tính. Để cho đơn giản, nên tập trung vào user là con người. Một role là một chức năng công việc hay tên công việc trong tổ chức theo thẩm quyền và trách nhiệm trao cho từng thành viên. Một permission là một sự cho phép của một chế độ cụ thể nào đó truy cập vào một hay nhiều object trong hệ thống. Các thuật ngữ authorization (sự trao quyền), access right (quyền truy cập) và privilege (quyền ưu tiên) đều để chỉ một permission. Các permission luôn tích cực và trao cho người có permission khả năng thực hiện một vài công việc trong hệ thống. Các object là các số liệu object cũng như là các nguồn object được thể hiện bằng số liệu trong hệ thống máy tính. Mô hình chấp nhận một loạt các cách diễn giải khác nhau cho các permission.

Role có cấp bậc

Mô hình RBAC1 giới thiệu role có thứ bậc (Role Hierarchies - RH). Role có thứ bậc cũng được cài đặt trong hệ thống tương tự như các role không thứ bậc. Role có thứ bậc có một ngữ nghĩa tự nhiên cho các role có cấu trúc để phản ánh một tổ chức của các permission và trách nhiệm. Trong một số hệ thống hiệu quả của các role riêng tư đạt được bởi khối bên trên thừa kế của các permission. Trong một số

trường hợp của hệ thống thứ bậc không mô tả sự phân phối của permission chính xác. Điều này thích hợp để giới thiệu các role riêng tư và giữ ngữ nghĩa của hệ thống thứ bậc liên quan xung quanh những role không thay đổi.

Các ràng buộc

Các ràng buộc là một thành phần quan trọng của RBAC và được cho là có tác dụng thúc đẩy sựphát triển của RBAC. Các ràng buộc trong RBAC có thể được áp dụng cho các quan hệgiữa UA, PA, user và các chức năng của role trong với các session khác nhau. Các ràng buộc được áp dụng tới các quan hệ và các chức năng, sẽ trả về một giá trị có thể chấp nhận được hay không thể chấp nhận được. Các ràng buộc có thể được xem như các câu trong một vài ngôn ngữ chính thức thích hợp.

Mô hình hợp nhất

RBAC3 là sự kết hợp của RBAC1 và RBAC2 để cung cấp cả hai hệ thống thứ bậc role và các ràng buộc. Có một số vấn đề xảy ra khi kết hợp hai môn hình trong một hệ thống thống nhất. Các ràng buộc có thể được áp dụng cho các hệ thống role có thứ bậc. Hệ thống role thứ bậc được yêu cầu tách nhỏ ra từng phần.

Các ràng buộc là cốt lõi của mô hình RBAC3. Việc thêm các ràng buộc có thể giới hạn số các role của người cấp cao (hay người cấp thấp) có thể có. Hai hay nhiều role có thể được ràng buộc để không có sự phổ biến role của người cấp cao (hay người cấp thấp). Các loại ràng buộc này là hữu ích trong hoàn cảnh mà việc xác thực thay đổi vai trò hệ thống role có thứ bậc được chuyển giao, nhưng trưởng security officer chuyển toàn bộ các loại trong thay đổi được thực hiện.

Các mô hình quản lý

Các ràng buộc được áp dụng tới tất cả các thành phần. Các role quản lý AR và các quyền quản lý AP được tách biệt ra giữa các role thông thường R và các permission P. Mô hình hiển thị các permission có thể được gán tới các role và các permission quản lý có thể chỉ được gán tới các role quản lý. Điều này gắn liền các ràng buộc.

3.2.4 Điều khiển truy cập dựa trên luật (Rule BAC– Rule Based Access Control)

Kiểm soát truy nhập dựa trên luật cho phép người dùng truy nhập vào hệ thống vào thông tin dựa trên các luật (rules) đã được định nghĩa trước.

Firewalls/Proxies là ví dụ điển hình về kiểm soát truy nhập dựa trên luật:

- Dựa trên địa chỉ IP nguồn và đích của các gói tin.
- Dựa trên phần mở rộng các files để lọc các mã độc hại.
- Dựa trên IP hoặc các tên miền để lọc/chặn các website bịcấm.
- Dựa trên tập các từ khoá để lọc các nội dung bị cấm.

Chương 4: Điều khiển truy xuất dữ liệu lớn

4.1 Giới thiệu

Dữ liệu chỉ số IQ [3] News ước tính rằng dân số dữ liệu toàn cầu sẽ đạt 44 zettabyte (1 tỷ terabyte) vào năm 2020. Điều này có xu hướng gia tăng ảnh hưởng tới cách thu thập hàng loạt dữ liệu và tính toán tốc độ cao hoặc hoạt động và phân tích kế hoạch. Big Data (BD) đề cập đến lớn dữ liệu đó là khó khăn cho quá trình bằng cách sử dụng hệ thống xử lý dữ liệu truyền thống, ví dụ, để phân tích lưu lượng truy cập dữ liệu Internet, hoặc chỉnh sửa video dữ liệu của hàng trăm gigabyte (Lưu ý rằng mỗi trường hợp phụ thuộc vào khả năng của một hệ thống; nó đã được lập luận rằng đối với một số tổ chức, hàng terabyte các văn bản, âm thanh và video dữ liệu mỗi ngày có thể được xử lý, do đó, nó không phải là BD, nhưng đối với những tổ chức mà không thể xử lý một cách hiệu quả, nó là BD[16]). Công nghệ BD đang dần định hình lại hệ thống dữ liệu hiện tại và thực hành. Chính phủ máy tính News[13] ước tính rằng khối lượng dữ liệu lưu trữ của các cơ quan của liên bang sẽ tăng 1,6-2,6 petabytes trong vòng hai năm, và tiểu bang Hoa Kỳ và chính quyền địa phương chỉ là quan tâm đến khai thác sức mạnh của BD để tăng cường an ninh, ngăn chặn gian lận, tăng cường cung cấp dịch vụ, và cải thiện phản ứng khẩn cấp. Người ta ước tính rằng tận dụng thành công công nghệ cho BD có thể làm giảm IT chi phí trung bình 48%[22]. BD có độ phân giải đặc hơn và cao hơn như phương tiện truyền thông, hình ảnh, và video từ các nguồn như phương tiện truyền thông xã hội, điện thoại di động ứng dụng, hồ sơ công cộng và cơ sở dữ liệu; các dữ liệu hoặc là trong lô tĩnh hoặc động bằng máy và người sử dụng bởi khả năng tiên tiến của phần cứng, phần mềm và mạng công nghệ. Các ví dụ bao gồm dữ liệu từ các mạng cảm biến hoặc theo dõi hành vi người dùng. Tăng nhanh khối lượng dữ liệu và đối tượng dữ liệu thêm áp lực rất lớn về CNTT hiện tại cơ sở hạ tầng với những khó khăn như khả năng mở rộng quy mô cho lưu trữ dữ liệu, phân tích trước, và bảo mật. Những khó khăn kết quả từ các tập tin lớn và ngày càng tăng của BD với tốc độ cao, và trong định dạng khác nhau, như được đo bằng: vận tốc - Velocity (các dữ liệu đi kèm tốc độ cao, ví dụ, dữ liệu khoa học như dữ liệu từ thời tiết các mẫu), khối lượng - Volume (kết quả dữ liệu

từ các tập tin lớn, ví dụ, Facebook đã tạo 25TB dữ liệu hàng ngày), và đa dạng (các tập tin đến trong các định dạng khác nhau: âm thanh, video, tin nhắn văn bản, vv [16]). Do đó, hệ thống xử lý dữ liệu BD phải có khả năng để đối phó với việc thu thập, phân tích, và đảm bảo dữ liệu BD mà đòi hỏi xử lý tập dữ liệu rất lớn mà bất chấp dữ liệu thông thường công nghệ quản lý, phân tích, và an ninh. Một cách đơn, một số giải pháp sử dụng một hệ thống riêng xử lý cho BD. Tuy nhiên, để tối đa hóa khả năng mở rộng và thực hiện, hầu hết các hệ thống xử lý BD áp dụng ồ ạt phần mềm chạy song song trên nhiều máy tính trong khung phân phối máy tính mà có thể bao gồm cột cơ sở dữ liệu và các giải pháp quản lý BD khác[15].

Hệ thống điều khiển truy xuất - Access Control (AC) là một trong những các thành phần an ninh mạng quan trọng nhất. Có nhiều khả năng rằng sự riêng tư hoặc an ninh sẽ bị tổn hại do các sai của chính sách kiểm soát truy cập hơn từ một thất bại của một mật mã nguyên thủy hoặc các giao thức. Vấn đề này càng trở nên nghiêm trọng như các hệ thống phần mềm ngày càng trở nên phức tạp chẳng hạn như hệ thống xử lý BD, được triển khai để quản lý một số lượng lớn các thông tin nhạy cảm và tài nguyên tổ chức thành một cụm chế biến tinh vi BD. Về cơ bản, hệ thống AC BD đòi hỏi sự hợp tác giữa các công ty xử lý bảo vệ môi trường máy tính, trong đó bao gồm các đơn vị tính toán theo phân phối quản lý AC[14].

Nhiều thiết kế kiến trúc đã được đề xuất để giải quyết thách thức BD; Tuy nhiên, hầu hết trong số đã được tập trung vào khả năng xử lý vận tốc (Velocity), khối lượng (Volume), và đa dạng (Variety). Cân nhắc cho an ninh trong việc bảo vệ BD AC chủ yếu là quảng cáo và những nỗ lực vá. Ngay cả với một số khả năng bảo mật trong các hệ thống BD gần đây, AC thực tế (ủy quyền) cho các thành phần xử lý BD là không có sẵn. Phần này đề xuất một kế hoạch tổng quát của AC phân phối cụm xử lý BD, Ứng dụng Search Engine phân tán trên nền tảng Hadoop-Nutch .

4.2 Nutch - Ứng dụng Search Engine phân tán trên nền tảng Hadoop

4.2.1 Ngữ cảnh ra đời và lịch sử phát triển của Nutch

Nutch là một dự án phi lợi nhuận của apache nhằm phát triển một ứng dụng search engine nguồn mở hoàn chỉnh bằng ngôn ngữ Java. Nutch được Dough

Cutting khởi xướng vào năm 2002. Cho đến nay, Nutch đã được khá nhiều công ty, tổ chức sử dụng để xây dựng nên các bộ máy tìm kiếm riêng cho mình. Sau đây là một số điểm chính trong quá trình hình thành và phát triển Nutch:

Năm 2002, Nutch ra đời dưới sự khởi xướng của Dough Cutting và Mike Cafarella. Dough Cutting cũng là người khởi xướng hai dự án Hadoop và Lucene.

Năm 2003, hệ thống demo đã thực hiện thành công với 100 triệu trang web. Tuy nhiên, các nhà phát triển nhanh chóng nhận ra kiến trúc của họ không thể mở rộng ra để tìm kiếm trên hàng tỷ trang web.

Năm 2003, Google công bố kiến trúc hệ thống file phân tán GFS. Nutch nhanh chóng áp dụng kiến trúc này để xây dựng nên một hệ thống file phân tán NDFS riêng cho mình.

Năm 2004, Google công bố mô hình xử lý dữ liệu phân tán MapReduce. Các nhà phát triển Nutch đã lập tức áp dụng mô hình này. Đầu năm 2005, hầu các thuật toán của Nutch đã được chuyển qua MapReduce. Hệ thống search engine của Nutch đã có thể mở rộng ra để thực hiện tìm kiếm trên hàng tỷ trang web.

Tháng sáu năm 2005, Nutch chính thức gia nhập vào ngôi nhà nguồn mở Apache Software Foundation. Cho đến nay, các bản release mới của Nutch vẫn liên tục ra đời. Bản release mới nhất cho đến thời điểm tài liệu này được viết là Nutch 1.1, được công bố vào đầu tháng sáu năm 2010.

4.2.2 Giới thiệu Nutch

4.2.2.1 Nutch là gì?

Nutch là một dự án nguồn mở thuộc Apache Software Foundation với mục tiêu một ứng dụng web search engine hoàn chỉnh. Nutch được xây dựng trên nền tảng lưu trữ và tính toán phân tán, do vậy Nutch có thể được mở rộng để hoạt động trên các tập dữ liêu lớn. Nutch bao gồm:

- Một chương trình thu thập web (web crawler)
- Các Parser cho nội dung các tài liệu thu thập được.
- Một công cụ xây dựng đồ thị liên kết các trang web (link-graph).

- Các thành phần tạo chỉ mục tài liệu và tìm kiếm.
- Nền tảng tính toán phân tán, giúp cho Nutch có khả năng mở rộng cao.
- Kiến trúc plugin-based giúp cho Nutch có tính linh hoạt.

Nutch được cài đặt bằng Java. Do vậy Nutch có thể chạy trên nhiều hệ điều hành và phần cứng khác nhau. Nutch có thể chạy ở chế độ stand alone (chế độ chạy trên một máy đơn), tuy nhiên Nutch chỉ phát huy hết sức mạnh của nó nếu được chạy trên một Hadoop cluster.

4.2.2.2 Nền tảng phát triển

Việc xây dựng một ứng dụng Search Engine từ đầu đến cuối là một nhiệm vụ quá khó khăn. Vì vậy Dough Cutting và Mike Cafarella đã sử dụng một số nền tảng có sẵn để phát triển Nutch.

4.2.2.2.1 Lucene

Lucene là một bộ thư viện nguồn mở có chức năng xây dựng chỉ mục và tìm kiếm. Bản thân Lucene cũng là một dự án của Apache Software Foundation, được khởi xướng bởi Dough Cutting. Nutch sử dụng Lucene để thực hiện tạo chỉ mục cho các tài liệu thu thập được và áp dụng vào chức năng tìm kiếm, chấm điểm (ranking) của search engine.

4.2.2.2.2 Hadoop

Hadoop bắt nguồn từ Nutch (xem 2.4.1.2, Lịch sử Hadoop). Hadoop cung cấp cho Nutch nền tảng tính toán phân tán với một hệ thống tập tin phân tán (HDFS) và một framework để xây dựng các ứng dụng MapReduce (MapReduce engine).

4.2.2.2.3 Tika

Tika là một bộ công cụ dùng để phát hiện và rút trích các metadata và những nội dung văn bản có cấu trúc từ những loại tài liệu khác nhau. Nutch sử dụng Tika để phân tách các loại tài liệu khác nhau, nhằm sử dụng cho quá trình tạo chỉ mục.

4.2.2.3 Các tính năng của Nutch.

4.2.2.3.1 Hỗ trợ nhiều protocol

Hiện nay Nutch hỗ trợ các các protocol khác nhau như: http, ftp, file. Nhờ kiến trúc plugin-based, ta có thể dễ dàng mở rộng ra thêm các protocol cho Nutch (xem phần 4.2.4.2.2 , Protocol plugins)

4.2.2.3.2 Hỗ trợ các loại tài liệu khác nhau

Hiện nay, Nutch có thể phân tách (parse) khá nhiều loại tài liệu khác nhau:

- Plain text
- HTML
- XML
- JavaScript: không chỉ phân tách các link từ mã js như các search engine khác mà còn phân tách cả phần code của js.
- Open Office (OpenOfice.org): phân tách tài liệu Open Office và Star Office.
- Tài liệu Microsoft: Word (.doc), Excel(.xls), Power Point(.ppt).
- Adobe PDF
- RSS
- Rich text format document (.rft)
- MP3: phân tách các trường D3v1 hay ID3v2 chứa các nội dung về bài hát như: tựa đề, album, tác giả…
- ZIP: Mặc định Nutch chỉ xử lý hai loại tài liệu là HTML và Plain Text. Muốn mở rộng ra cho các tài liệu khác, ta phải cấu hình lại Nutch. Cũng nhờ vào kiến trúc Plugin-based của Nutch mà ta có thể viết thêm các Plugin để xử lý các tài liệu khác (xem phần 4.2.4.2.3)

4.2.2.3.3 Chạy trên hệ thống phân tán

Nutch được phát triển trên nền tảng Hadoop. Do vậy ứng dụng Nutch sẽ phát huy được sức mạnh của nó nếu chạy trên một Hadoop cluster. Điều này mang lại cho Nutch một khả năng mở rộng (dựa trên khả năng mở 4.2.4.2 của Hadoop cluster) để có thể trở thành một search engine thực thụ (xem 4.2.5).

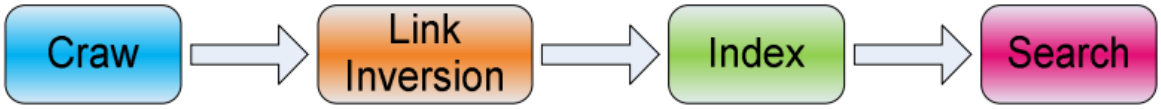
4.2.2.3.4 Tính linh hoạt với plugin

Nhờ kiến trúc plugin-based, ta có thể dễ dàng mở rộng các tính năng của Nutch bằng cách phát triển thêm các plugin. Tất cả các thao tác phân tách tài liệu, lập chỉ mục và tìm kiếm đều thật sự được thực hiện bởi các plugin (xem 4.2.4.2).

4.2.3 Kiến trúc ứng dụng Nutch

4.2.3.1 Thuật giải Nutch

Quá trình hoạt động của Nutch thường trải qua các giai đoạn chính sau đây là thu thập tài liệu (crawl), đảo ngược các link (link inversion), tạo chỉ mục (index) và tìm kiếm.



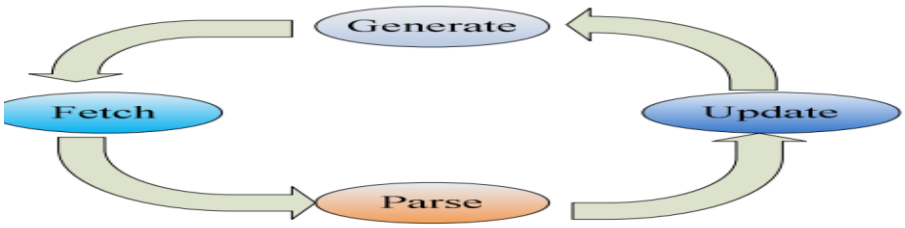
Hình 3-1 Các bước chính trong thuật toán của Nutch

4.2.3.1.1 Thu thập tài liệu (crawling)

Dữ liệu trạng thái của quá trình crawling được lưu trữ trong một cấu trúc dữ liệu có tên crawldb (chi tiết ở 4.2.3.2.1). CrawlDB chứa tất cả các URL được crawl và các thông tin đi kèm như ngày nạp, tình trạng nạp (chưa nạp, đã nạp, đã nạp nhưng đã lỗi thời…) và số lượng liên kết đến URL này. CrawlDB được khởi động bằng việc tiêm một số các URL hạt nhân.

Quá trình crawl sau khi được kích hoạt sẽ diễn ra theo một vòng lặp sau:

1) Generate: phát sinh các danh sách URL để nạp từ crawldb.
2) Fetch: nạp các URL từ danh sách nạp.
3) Parse: phân tách các tài liệu nạp được.
4) Update database (cập nhật lại crawldb): cập nhật trạng thái các URL được nạp, và thêm vào các URL mới tìm thấy từ kết quả của quá trình phân tách.



Hình 3-2 Vòng lặp crawl

4.2.3.1.1.1. Generate

Một danh sách các URL sẽ được phát sinh ra để dùng có quá trình nạp. Các URL được chọn là các URL trong crawldb có tình tr ạng là "chưa nạp" hoặc "đã lỗi thời". Số lượng URL được phát sinh ra trong danh sách có thể bị giới hạn lại theo ý muốn.

4.2.3.1.1.2. Fetch

Một trình tự động (fetcher) tự động tải các tài liệu từ danh sách URL được phát sinh ra từ bước generate. Fetcher có khả năng nạp các tài liệu với những protocol khác nhau như : http, ftp, file... Kết quả của quá trình này là dữ liệu nhị phân nạp được ứng với mỗi URL trong danh sách nạp.

4.2.3.1.1.3. Parse

Dữ liệu nhị phân có được từ giai đoạn nạp sẽ được phân tách ra để trích lấy các thông tin như các link, metadata và các thông tin văn bản khác để dùng cho việc tạo chỉ mục ngược sau này.

4.2.3.1.1.4. Update

Trạng thái của mỗi URL được nạp cũng như danh sách các URL mới có được từ quá trình parse đều sẽ được trộn vào phiên bản trước của crawldb để tạo ra một phiên bản mới. Các URL được nạp thành công/thất bại sẽ được cập nhật trạng thái tương ứng, số lượng link đi vào mỗi URL sẽ được cập nhật, và các URL mới sẽ được thêm vào crawldb như là một record mới.

4.2.3.1.2 Đảo ngược liên kết (link inversion)

Tất cả các URL trong crawldb sẽ được xử lý bằng duy nhất một thao tác MapReduce để phát sinh ra với mỗi URL một tập hợp các link trỏ vào URL đó (cả anchor text lẫn URL nguồn). Các anchor text sẽ được dùng trong quá trình tạo chỉ mục và tìm kiếm nhằm gia tăng chất lượng cho kết quả tìm kiếm.

4.2.3.1.3 Tạo chỉ mục ngược

Một thao tác MapReduce sẽ được thực hiện để kết hợp tất cả các thông tin đã biết được với mỗi URL như: phần text của trang web, anchor text của các liên kết trỏ vào URL, tựa đề tài liệu, metadata… Các dữ liệu này sẽ được dùng làm đầu vào

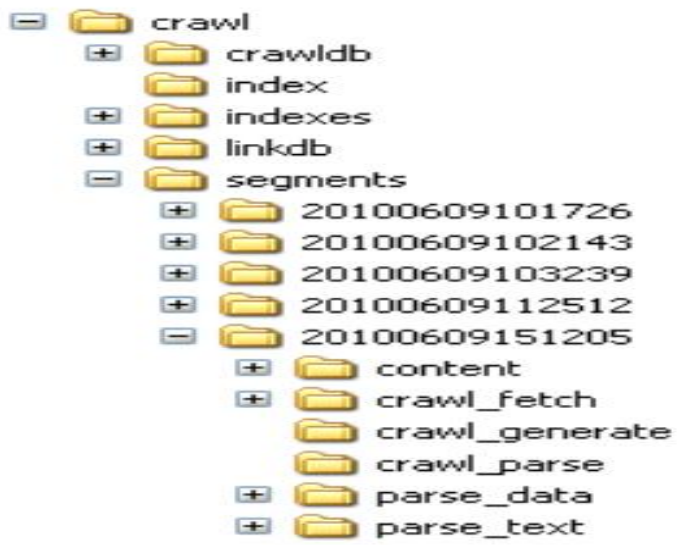
cho quá trình tạo ra các tài liệu Lucene và sau đó dùng các tài liệu này sẽ được tạo chỉ mục Lucene.

4.2.3.1.4 Tìm kiếm

Nutch được cài đặt trên hệ thống phân tán dưới nền tảng Hadoop. Tuy nhiên, không giống với các thuật giải khác, tìm kiếm không dùng tới MapReduce. Tập index sẽ được phân bổ xuống hệ thống local của các Search Node (Search server). Mỗi câu truy vấn tìm kiếm sẽ được gửi tới các Search node. Các Search node sẽ tìm kiếm cục bộ trên phần index của mình và trả kết quả về. Những kết quả được đánh giá cao nhất sẽ được hiển thị lên cho người dùng.

4.2.3.2 Cấu trúc dữ liệu của Nutch

Nutch sử dụng một vài khối dữ liệu, bao gồm crawldb, segments, indexes và linkdb. Chúng ta xem xét chi tiết cấu trúc các khối dữ liệu này:



Hình 3-3 Các thành phần dữ liệu Nutch

4.2.3.2.1 CrawlDb (crawl database)

CrawlDb chứa đựng tất cả các thông tin về trạng thái crawl của tất cả các URL mà Nutch biết. CrawlDb là một thư mục chứa các file, mỗi file chứa nhiều cặp giá trị <URL, CrawlDatum>.

CrawlDatum là một cấu trúc dữ liệu lưu trạng thái nạp của URL tương ứng, thông tin của một CrawlDatum gồm có:

modifiedTime: thời điểm cuối cùng thao tác với URL.

fetchTime: thời điểm nạp URL này.

linkCount: số lượng link trỏ đến URL này.

status: thông tin trạng thái nạp của URL. Có thể bao gồm các thông tin sau:

- db_unfetched: URL chưa được nạp.
- db_fetched: URL đã nạp.
- db_gone: đã nạp rồi nhưng đã lỗi thời (hết hạn).
- fetch_success: nạp thành công.
- fetch_fail: nạp thất bại.
- fetch_gone: quá trình nạp URL này kéo dài quá thời hạn (time out), nạp thất bại.

4.2.3.2.2 Segments

Segments là tập hợp chứa nhiều segment.

Mỗi segment chứa tất cả các tài liệu được thu thập tại mỗi vòng lập crawl (xem phần 4.2.3.1.1) và các dữ liệu tương ứng. Thông tin trên một segment gồm:

- Danh sách tất cả các URL được nạp cho segment.
- Dữ liệu mà fetcher tải được tương ứng với mỗi URL.
- Dữ liệu tải được sẽ được phân tách để trích lấy các trường thông tin text, các kết quả của quá trình này được lưu lại trong sengment.

Mỗi segment sẽ có một thời hạn, mặc định là 30 ngày. Quá thời hạn này dữ liệu trên segment coi như đã lỗi thời, cần phải được refetch (nạp lại). Thông thường người ta thường xoá đi các segment đã quá hạn để tiết kiệm đĩa cứng. Segment được đặt tên bằng chính thời điểm segment được tạo ra, theo format: theo fortmat:

ddMMyyhhmmss. Vì vậy ta có thể dễ dàng biết được một thời gian tồn tại của một segment.

Về mặt vật lý (lưu trữ), mỗi segment là một thư mực chứa các thư mục con sau đây:

- crawl_generate: chứa danh sách các URL sẽ được nạp.

 crawl_fetch: chứa trạng thái nạp của từng URL, sẽ được dùng để update lại CrawlDB.

 content: chứa nội dung thô, tức dữ liệu nhị phân tải được với từng URL.

 crawl_parse: chứa các outlink URL, dùng để update lại CrawlDB

 parse_data: chứa các oulink (các link từ tài liệu đi ra) và các metadata ứng với từng URL.

 parse_text: chứa tất cả text phân trích được với từng URL.

Tất cả các segment phát sinh ra trong quá trình crawl sẽ được lưu trữ trong segments.

4.2.3.2.3 Indexes

Phần indexes chứa tất cả chỉ mục ngược (inverted index) của tất cả các tài liệu mà hệ thống đã nhận được. Để tạo được phần indexes này, đầu tiên hệ thống sẽ tạo ra trên mỗi segment một tập index. Sau đó Nutch sẽ trộn tất cả phần index này lại để tạo ra indexes. Nutch sử dụng Lucene để tạo chỉ mục, do đó tất cả các tập chỉ mục trên Nutch đều theo cấu trúc của Lucene.

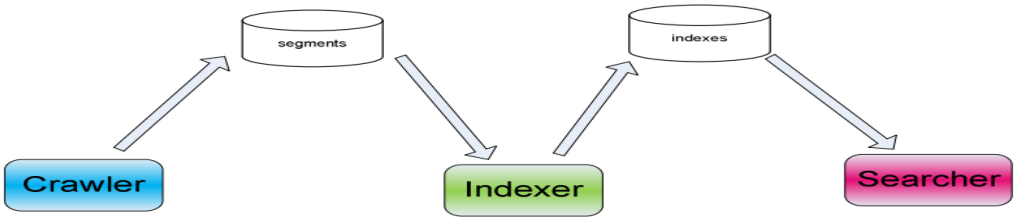
4.2.3.2.4 LinkDB (link database)

Với mỗi URL, LinkDb chứa tất cả các inlink vào URL bao gồm cả địa chỉ URL nguồn và anchor text. LinkDb được dùng vào quá trình tạo chỉ mục và đánh giá điểm (scoring) cho quá trình tìm kiếm.

4.2.4 Kiến trúc Nutch

4.2.4.1 Kiến trúc các thành phần

Kiến trúc của Nutch được phân chia một cách tự nhiên thành hai thành phần chính: crawler, indexer và searcher. Crawler thực hiện thu thập các tài liệu, phân tách các tài liệu, kết quả của crawler là một tập dữ liệu segments gồm nhiều segments. Indexer lấy dữ liệu do crawler tạo ra để tạo chỉ mục ngược. Searcher sẽ đáp ứng các truy vấn tìm kiếm từ người dùng dựa trên tập chỉ mục ngược do indexer tạo ra.



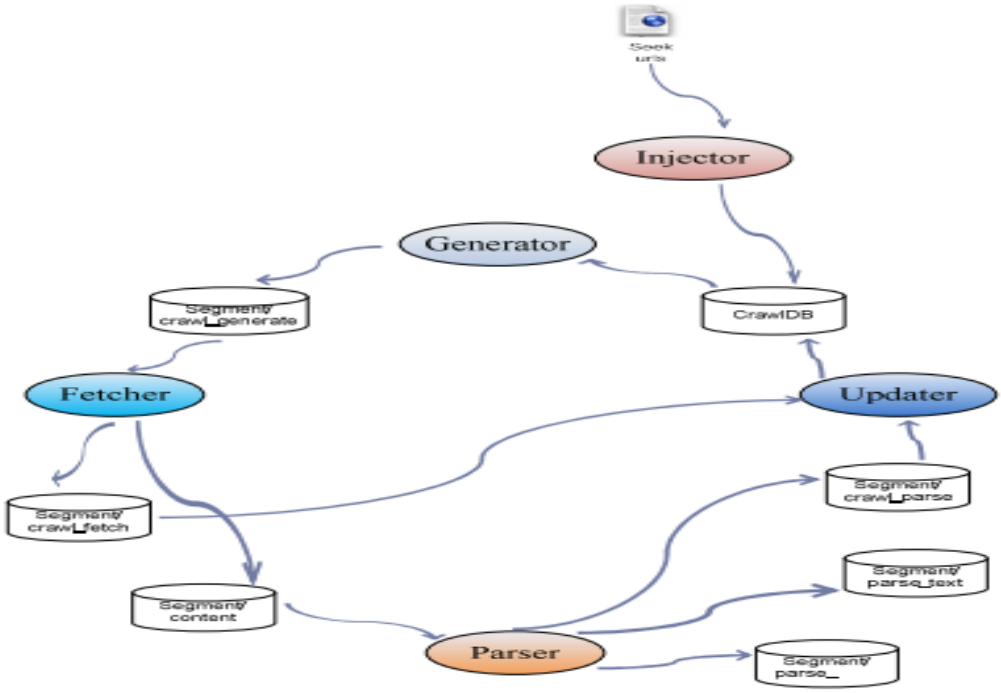
Hình 3-4 Tổng quan các thành phần của Nutch

Chi tiết các thành phần sẽ được mô tả chi tiết sau đây:

4.2.4.1.1 Crawler

Thành phần Crawler gồm các thành phần như hình sau:

Quá trình crawl được khởi động bằng việc module injector tiêm một danh sách các URL vao crawldb để khởi tạo crawldb.



Hình 3-5 Kiến trúc các thành phần và quá trình thực hiện crawler

Crawler gồm có bốn thành phần chính là generator, fetcher, parser và updater hoạt động liên tiếp nhau tạo thành một vòng lặp. Tại mỗi lần lặp, crawler sẽ tạo ra một segment.

Tại khởi điểm của vòng lặp, generator sẽ dò tìm trong crawldb các URL cần nạp và phát sinh ra một danh sách các url sẽ nạp. Đồng thời lúc này generator sẽ phát sinh ra một segment mới, lưu danh sách URL sẽ nạp vào segment/crawl_fetch.

Tiếp theo, fetcher sẽ lấy danh sách URL cần nạp từ segment/crawl_generate, thực hiện tải các tài liệu theo từng URL. Fetcher s ẽ lưu nội dung thô của từng tài liệu vào segment/content và lưu trạng thái nạp của từng URL vào segment/crawl_fetch.

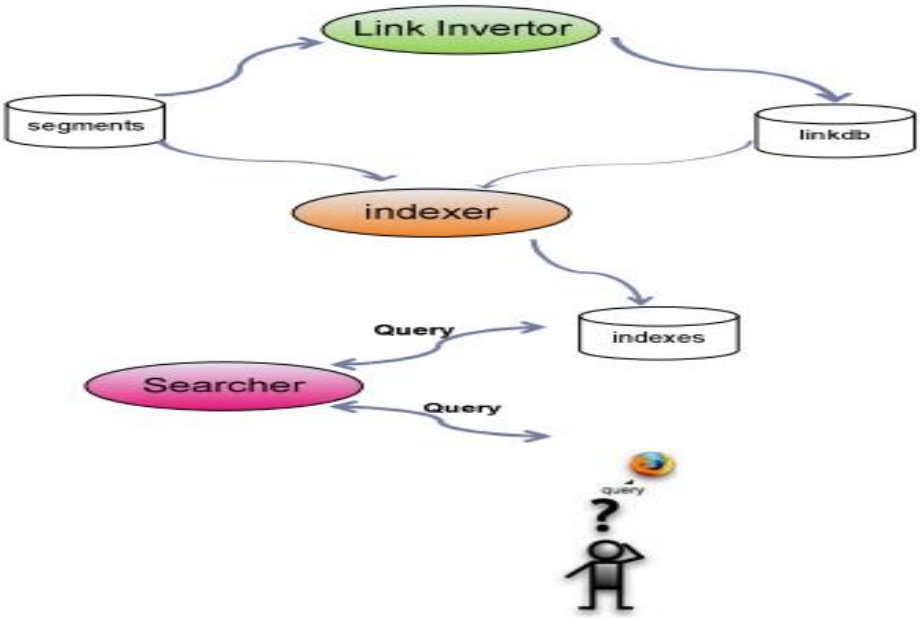
Sau đó, parser sẽ thực hiện lấy dữ liệu thô của các tài liệu từ segment/content và thực hiện phân tách các tài liệu để trích lấy các thông tin văn bản:

- Trích các outlink và lưu vào segment/crawl_parse
- Trích các outlink và metadata vào segment/parse_data
- Trích toàn bộ phần text của tài liệu vào segment/parse_text

Cuối cùng, crawldb sẽ sử dụng thông tin về các trạng thái nạp của từng URL trong segment/crawl_fetch và danh sách các URL mới phân tách được trong segment/crawl_parse để cập nhật lại crawldb.

Quá trình trên được lặp đi lặp lại. Số lần lặp của vòng lặp này được gọi là độ sâu (depth).

4.2.4.1.2 Indexer và Searcher



Hình 3-6 Các thành phần và quá trình thực hiện index và search

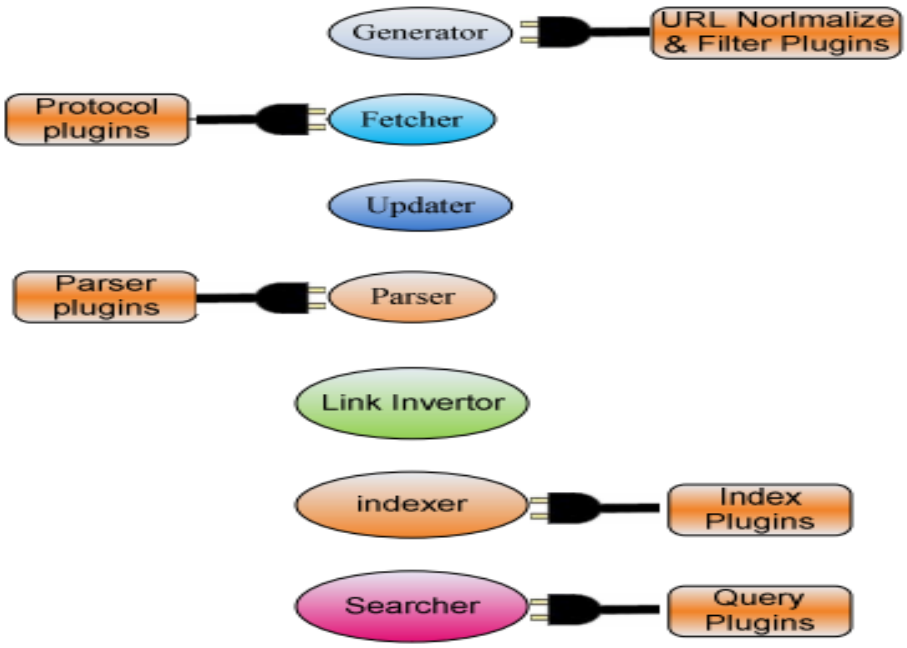
Link Invertor sẽ lấy dữ liệu từ tất cả các segment để xây dựng linkdb. Linkdb chứa tất cả các URL mà hệ thống biết cùng với các inlink của chúng (xem cấu trúc của linkdb tại 4.2.3.2.4)

Với từng segment trong segments, indexer sẽ tạo chỉ mục ngược cho segments. Sau đó, nó sẽ thực hiện trọn tất cả các phần chỉ mục này lại với nhau trong indexes.

User của hệ thống sẽ tương tác với các chương trình tìm kiếm phía client. Bản thân Nutch cũng đã hỗ trợ sẵn một ứng dụng web để thực hiện tìm kiếm. Các chương trình phía client này nhận các query từ người dùng, gửi đến searcher. Searcher thực hiện tìm kiếm trên tập chỉ mục và gửi trả kết quả lại cho chương trình phía client để hiển thị kết quả ra cho người dùng.

4.2.4.2 Plugin-based

Hầu hết các thành phần của Nutch đều sử dụng các plugin để thực hiện các chức năng của mình. Điều này làm cho các tính năng của Nutch có thể dễ dàng được mở rộng bằng cách thêm vào các plugin. Hình 3-7 cho thấy toàn cảnh các thành phần sử dụng plugin của Nutch.



Hình 3-7: Plugin trong Nutch

Sau đây ta sẽ xem xét các giao diện chức năng từng plugin

4.2.4.2.1 URL Norlmalize và Filter Plugins

Các plugin này được gọi khi có một URL mới được đưa vào hệ thống. Một plugin Normalizer sẽ thực hiện chuẩn hoá các URL thành một dạng tiêu chuẩn nhằm dễ dàng so sánh các URL và tránh được các lỗi URL không hợp lệ. Các thao tác chuẩn hoá như chuyển tất cả sang dạng viết thường (lower case), loại bỏ các chỉ port mặc định (ví dụ như port 80 cho protocol http).

Một plugin Filter sẽ làm nhiệm vụ quyết định xem có cho phép một URL được vào hệ thống hay không. Một filter plugin sẽ được sử dụng để giới hạn việc crawling trong một domain nào đó, để có thể crawling trong một intranet hay một miền nào đó có internet. Các filter plugin hiện có của Nutch sử dụng regular expresstion để lọc các URL, chia làm hai loại: White list và black list.

4.2.4.2.2 Protocol plugins

Mỗi một protocol plugin sẽ thực hiện nhiệm vụ tải nội dung của tài liệu từ một URL với một protocol nào đó. Ta có thể có plugin chuyên tải các URL HTTP, plugin tải URL FTP, plugin tải URL File… Các plugin này được sử dụng trong quá tình nạp các tài liệu. Chúng ta có thể dễ dàng mở rộng các protocol mà Nutch có thể hoạt động bằng cách phát triển và đăng thêm các protocol plugin để tải dữ liệu theo một protocol nào đó.

4.2.4.2.3 Parser plugins

Từ dữ liệu thô có được từ các protocol plugin, các parser plugin có nhiệm vụ phân tách dữ liệu của tài liệu như text, link hay metadata…của một loại tài liệu nào đó. Các plugin này được dùng bởi parser.

Nutch đã xây dựng sẵn các parser plugin khác nhau cho các định dạng nh ư PDF, Word, Exel, RTF, HTML, XML…

4.2.4.2.4 Index plugins và query plugins

Nutch sử dụng Lucene cho việc tạo chỉ mục và tìm kiếm. Khi tạo chỉ mục, mỗi tài liệu đã đươc phân tách sẽ được gửi đến cho các plugin index để thực hiện tạo các

tài liệu Lucene và phát sinh chỉ mục. Các plugin index sẽ quyết định xem trường dữ liệu nào được tạo chỉ mục và sẽ tạo như thế nào.

Các câu truy vấn tìm kiếm trong Nutch được phân tách thành một cây truy vấn. Sau đó cây truy vấn này sẽ được gửi đến cho các query plugin, các plugin này sẽ phát sinh ra một Lucene query để có thể thực thi.

4.2.5 Nutch và việc áp dụng tính toán phân tán với mô hình MapReduce vào Nutch

4.2.5.1 Nguyên nhân cần phải phân tán

Hai khó khăn chính đặt ra cho các Search Engine như sau:

Khó khăn về lưu trữ: Do số lượng và kích thước các trang web trên internet tăng nhanh, nên khối lượng dữ liệu cần để lưu trữ cho các quá trình của search engine là quá lớn.

Khó khăn về xử lý: cũng do khối lượng dữ liệu cần xử lý là quá lớn, nên việc xử lý một cách tuần tự mất quá nhiều thời gian.

Hai khó khăn trên cũng là các khó khăn của Nutch ở giai đoạn đầu. Nhận thức được việc Nutch không thể mở rộng ra để thực hiện tìm kiếm trên hàng tỷ trang web. Dough Cutting đã áp dụng mô hình tính toán phân tán với MapReduce vào Nutch.

Hiện nay, Nutch sử dụng nền tảng phân tán với Hadoop. Hadoop cung cấp cho Nutch hệ thống tập tin phân tán HDFS và framework phát triển ứng dụng MapReduce.

4.2.5.2 Áp dụng tính toán phân tán cho các thành phần Crawler

Nutch đã chuyển tất cả các thuật toán trong quá trình crawl và tạo chỉ mục sang MapReduce: inject, generate, fetch, parse, updateDb, invert links, va index. Tất cả các kết quả lưu trữ, các khối dữ liệu của quá trình crawl và index như linkdb, crawldb, segments và indexes đều được lưu trữ trên hệ thống HDFS.

4.2.5.3 Áp dụng tính toán phân tán cho Searcher

Một hướng tự nhiên mà ai cũng nghĩ tới là chúng ta cũng sẽ sử dụng MapReduce để tìm kiếm phân tán trên tập index lưu trữ trên HDFS. Tuy nhiên, điều này mang lại một số bất lợi mà ta sẽ cùng khảo sát dưới sây. Sau đó, chúng tôi sẽ giới thiệu một giải pháp thật sự cho vấn đề tìm kiếm phân tán.

4.5.3.1 Hạn chế của việc sử dụng MapReduce để tìm kiếm chỉ mục trên HDFS

Việc sử dụng MapReduce để tìm kiếm tập chỉ mục trên HDFS sẽ diễn ra theo mô hình sau:

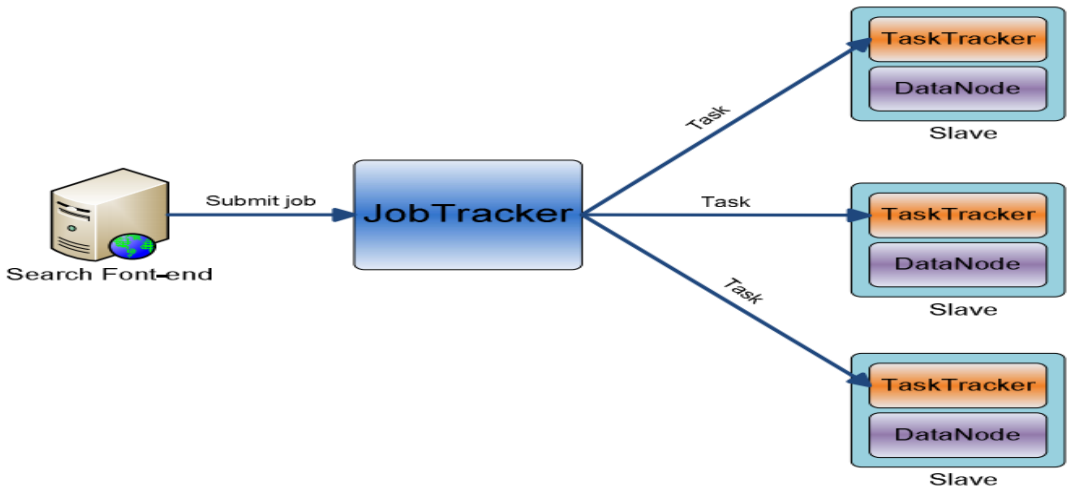
Đầu tiên, một trình Search Font-End nào đó sẽ nhận truy vấn từ người dùng và khởi tạo một MapReduce Job (xem lại phần 2.4.3.2.2.2. , cơ chế hoạt động MapReduce Engine) để gửi đến JobTracker thực thi. JobTracker sẽ phân Job này ra thành nhiều Map task và một Reduce task. Các task này sẽ được phân công cho các TaskTracker, TaskTracker sẽ khởi tạo và thực thi các task này.

Mỗi Map Task này sẽ thực hiện tìm kiếm trên 1 phần split tập chỉ mục, dò tìm trên phần split này các khớp với query tìm kiếm. Nhờ cơ chế locality nên thông thường phần split này sẽ nằm cùng máy vật lý với Task thực thi.

Reduce Task duy nhất sẽ thực hiện tổng hợp kết quả và ghi ra một file output trên HDFS.

Quá trình trên vấp phải một số bất tiện sau:

Việc xử lý một MapReduce Job đi theo quá trình sau: JobTracker nhận MapReduce job từ client (ở đây là các yêu cầu truy vấn) Search Font-End và đưa vào một hàng đợi các job để chờ thực thi. Tới lượt thực thi, JobTracker sẽ phân Job này ra các task và giao cho các TaskTracker xử lý. TaskTracker sẽ khởi tạo và thực thi các task. Quá trình này sẽ tạo ra một độ trễ do thời gian chờ job nằm trong hàng đợi, thời gian khởi tạo các task. Độ trễ này tuỳ theo phần cứng cụ thể mà mất khoảng vài giây hoặc lâu hơn. Độ trễ này không có nhiều ý nghĩa với các MapReduce job thông thường (là các job mà thời gian xử lý lên đến hàng giờ, hàng ngày, hàng tuần). Tuy nhiên, chúng ta đều biết thời gian thực thi truy vấn với Search Eninge có ý nghĩa tới từng giây. Độ trễ này sẽ ảnh hưởng nghiêm trọng tới thời gian thực thi truy vấn và làm cho thời gian đáp ứng của truy vấn không còn chấp nhận được nữa.



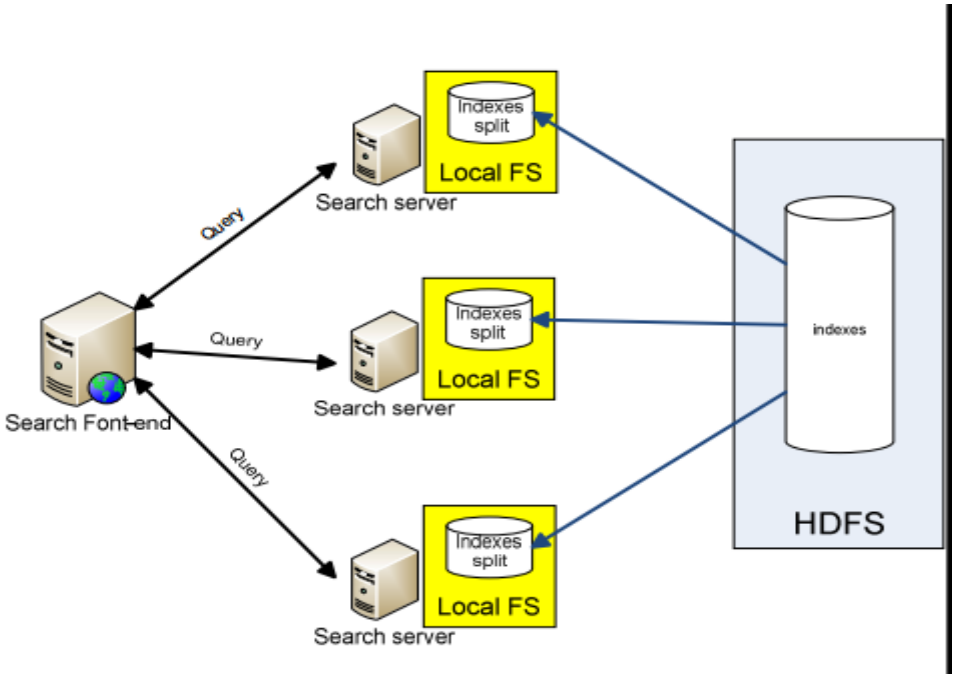
Hình 3-8: Quá trình truy vấn chỉ mục bằng MapReduce

Thứ hai, trên hệ thống, crawler luôn chạy. Và lúc đó, việc chạy Searher bằng MapReduce trên cùng cluster với crawler sẽ gây ra tình trạng tranh giành tài nguyên: tài nguyên băng thông khi các task truy vấn và các task cho quá trình crawl cùng thực hiện. Tài nguyên RAM và CPU trên các TaskTracker khi TaskTracker phải thực hiện nhiều task cùng một lúc. Đương nhiên điều này sẽ làm giảm đi thời gian thực thi của cả task thực hiện tìm kiếm và task thực hiện query.

Với những bất tiện kể trên, thì Nutch đã không khuyến kích việc tìm kiếm phân tán tập index bằng MapReduce trên HDFS. Thực chất, Nutch không đã cài đặt chức năng này. Thay vào đó, Nutch đưa ra một giải pháp tiềm kiếm phân tán khác. Đó là các Search server.

4.2.5.3.2 Search server

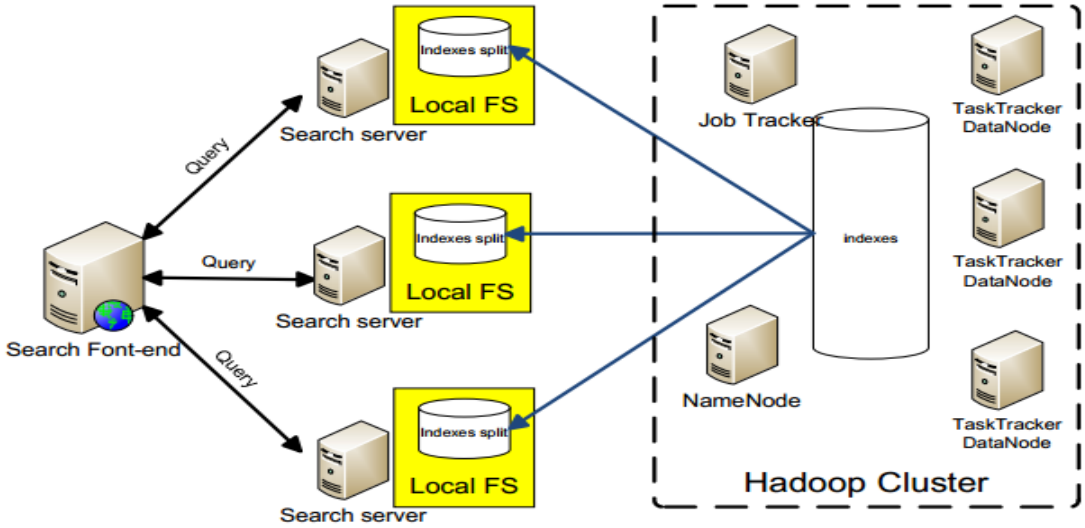
Với giải pháp này, tập chỉ mục trên HDFS sẽ được bổ ra thành nhiều phần nhỏ, mỗi phần sẽ lưu trữ trên một server chuyên phục vụ tìm kiếm: Search server. Các Search server này sẽ không nằm trong Hadoop cluster mà sẽ chạy độc lập. Các Search server lưu trữ phần index của nó trên hệ thống tập tin cục bộ của nó. Với ưu điểm của hệ thống file cục bộ là độ trễ (latency) thấp, các Search server sẽ đáp ứng yêu cầu tìm kiếm rất nhanh. Khi một trình Search Font-End nhận được truy vấn từ người dùng, nó sẽ gửi truy vấn này đến tất cả các search server, nhận kết quả trả về từ các Search server và tổng hợp kết quả để hiển thị cho người dùng.



Hình 3-9 Search servers

4.2.5.4 Mô hình ứng dụng Search Engine phân tán hoàn chỉnh

Từ các kết luận ở các phần trên, giờ đã đến lúc nhìn lại kiến trúc thật sự của một hệ thống Search Engine phân tán với Nutch.



Hình 3-10: Mô hình ứng dụng Search Engine phân tán hoàn chỉnh

Theo mô hình này, tất cả các công đoạn crawl, index sử dụng MapReduce và được thực hiện trên Hadoop cluster. Các kết quả của giai đoạn crawl và index cũng được lưu trên HDFS.

Kết quả của quá trình crawl và index là một khối dữ liệu chỉ mục được lưu trữ trên HDFS. Các khối này sẽ được phân bổ xuống các Search server nằm ngoài cluster.

Phần Phu lục E - Hướng dẫn triển khai hệ thống Search Engine phân tán Nutch sẽ mô tả chi tiết hơn và cách triển khai mô hình này.

Chương 5: Thực nghiệm và các kết quả

5.1 Giới thiệu

Trong chương này nhóm sẽ trình bày các thực nghiệm về triển khai ứng dụng Nutch. Mục đích chung của các thực nghiệm này là để khẳng định lại tác dụng của việc chạy Nutch trên môi trường phân tán Hadoop (HDFS và MapReduce).

Như ta đã biết, qui trình của Nutch trả qua hai giai đoạn chính:

- Crawl và tạo chỉ mục.
- Tìm kiếm trên tập chỉ mục.

Trong đó, như đã trình bày ở Chương 3, quá trình crawl được thực hiện phân tán bằng các thuật giải MapReduce và lưu kết quả ra HDFS. Quá trình Search thì thực hiện phân tán thông qua các Search server. Chúng ta sẽ lần lược tiến hành thực nghiệm cả hai quá trình này để thấy rõ tác dụng của việc phân tán.

5.2 Thực nghiệm triển khai crawl và tạo chỉ mục.

5.2.1 Mục đích

Như ta đã biết Nutch có thể chạy ở hai chế độ:

Stand alone: Nutch chạy trên một máy đơn, tất cả các quá trình được thực hiện trên một máy, dữ liệu lưu trữ ra hệ thống tập tin cục bộ của máy.

Distributed: Nutch chạy trên một Hadoop cluster. Tất cả các quá trình như fetch, generate, index… đều được thực hiện phân tán với MapReduce. Dữ liệu kết quả được lưu trên hệ thống tập tin phân tán HDFS.

Mục tiêu của thực nghiệm này là đánh giá được tác dụng của việc áp dụng tính toán phân tán Hadoop vào crawler. Điều này được thực hiện thông qua việc so sánh tốc độ thực hiện crawler ở hai chế độ Stand alone và Distributed.

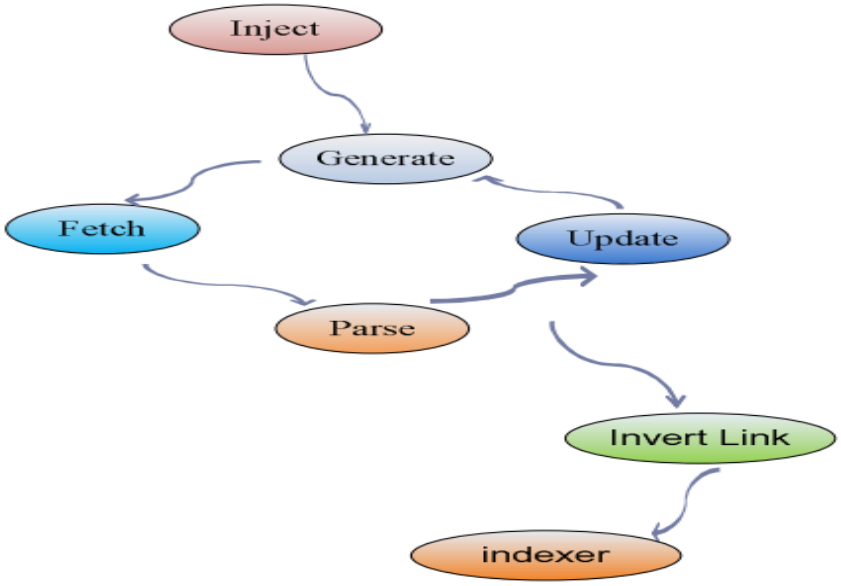
5.2.2 Phần cứng

Phần cứng thực hiện thực nghiệm gồm có ba máy:

- 1 máy (core2due, 2GB DDR2, 160GB HDD): is-teacher02.
- 2 máy (Pentium 4, 512MB RDRam, 40GB HDD): is-aupelf04, is-teacher06.

5.2.3 Phương pháp thực hiện

Quá trình crawler trải qua các giai đoạn sau:



Hình 4-1: Quy trình crawler

Ta tiến hành tạo một đoạn chương trình ngắn, để thực hiện tuần tự từ công việc và tiến hành do thời gian của từng giai đoạn. Mã giã của phần code để đo thời gian như sau:

start-time = now()

Thực hiện Inject

after_inject_time = now()

inject_duration = after_inject_time-start_time

for (i=1; i<=depth; i++)

{

Ghi nhận thông tin cho depth=i

before_generate_time=now()

Thực hiện Generate

after_generate_time=now()

generate_duration=after_generate_time – before_generate_time

Thực hiện Fetch

after_fetch_time = now()

fetch_duration = after_fetch_time - after_generate_time

Thực hiện Parse

after_parse_time = now()

parse_duration = after_parse_time – after_fetch_time

Thực hiện Update CrawlDB

after_update_time = now()

update_duration = after_update_time – after_parse_time

Ghi nhận tổng số URL đã nạp bà parser

}

before_invert_time = now()

Thực hiện Invert link

after_invert_time = now()

invert_duration = after_invert_time – before_invert_time

Thực hiện index

after_index_time = now()

index_duration = after_index_time – after_invert_time

total_duration = after_index_time – start_time

Thực hiện crawl với các điều kiện như sau:

Điều kiện	Giá trị	Mô tả
Seek URLs	http://hcm.24h.com.vn	Khởi động URL gốc
Chiều sâu	3	Lặp lại crawl loop 3 lần
URL Filter	+^http://([a-z)-9]*\.)*24h.com.vn/	Chỉ chấp nhận các URL thuộc miền 24h.com.vn

Bảng 1: Các điều kiện thực nghiệm crawl

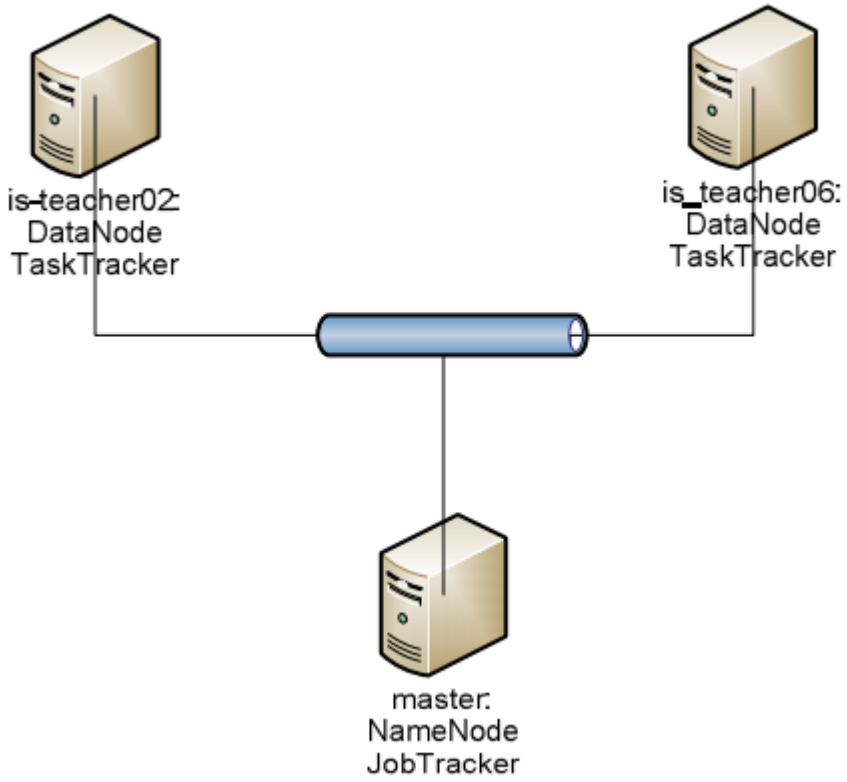
Ta thực hiện chạy chương trình này trên hai môi trường như mô tả dưới đây. Các kết quả về thời gian được lưu ra một file log để thực hiện so sánh, đánh giá.

5.2.3.1 Stand alone

Chạy hệ thống ở chế độ Stand alone trên máy is_aupelf04, cấu hình phần cứng như đã mô tả ở phần trên.

5.2.3.2 Distributed

Chạy hệ thống trên một Hadoop cluster như sau:



Hình 4-2: Mô hình thực nghiệm phân tán crawler

5.2.4 Kết quả

Kết quả thống kê thời gian thực hiện các quá trình như sau (Các khoảng thời gian nhanh hơn sẽ được in nghiêng):

Quy trình thực hiện			Stand alone (giây) (1)	Distributed(giây) (2)	Tỷ lệ % (2)/(1)
Inject			33	180	545%
Vòng lặp Crawl	Dept=1	Generate	54	256	474%
		Fetch	150	345	230%
	(Số URL=1)	Parse	210	260	124%
		Update DB	157	340	217%
		Tổng	571	1201	210%
	Dept=2	Generate	183	301	164%

		Fetch	2112	1683	80%
	(Số URL=149)	Parse	1385	1079	78%
		Update DB	851	902	106%
		Tổng	4531	3965	88%
	Dept=3	Generate	2095	1982	95%
		Fetch	27365	18910	69%
	(Số URL=10390)	Parse	6307	3550	56%
		Update DB	2381	2053	86%
		Tổng	38184	26495	69%
Invert Link			2581	2275	88%
Index			3307	2557	77%
Tổng thời gian			49171	36673	75%

Bảng 2: Kết quả thống kê đánh giá thực nghiệm crawl ở chế độ standalone và Distributed.

Tuy nhiên do thời gian đo bằng giây khó đánh giá, ta sẽ chuyển sang một dạng trực quan hơn:

Quy trình thực hiện			**Stand alone (giây) (1)**	**Distributed(giây) (2)**	**Tỷ lệ % (2)/(1)**
Inject			*33 giây*	180 giây	545%
Vòng lặp Crawl	Dept=1	Generate	*54 giây*	256 giây	474%
		Fetch	*150 giây*	345 giây	230%
	(Số URL=1)	Parse	*210 giây*	260 giây	124%
		Update DB	*157 giây*	340 giây	217%
		Tổng	571 giây	1201 giây	210%
	Dept=2	Generate	*183 giây*	301 giây	164%
		Fetch	35 phút	*28 phút*	*80%*
	(Số URL=149)	Parse	23 phút	*18 phút*	*78%*
		Update DB	*14 phút*	15 phút	106%
		Tổng	1 giờ 16 phút	*1 giờ 6 phút*	*88%*
	Dept=3	Generate	35 phút	*33 phút*	*95%*
		Fetch	7 giờ 36 phút	*5 giờ 15 phút*	*69%*
	(Số URL=10390)	Parse	1 giờ 45 phút	*59 phút*	*56%*

	Update DB	40 phút	*34 phút*	*86%*
	Tổng	10 giờ 36 phút	*7 giờ 22 phút*	*69%*
Invert Link		43 phút	*38 phút*	*88%*
Index		55 phút	*41 phút*	*77%*
Tổng thời gian		12 giờ 16 phút	*10 giờ 11 phút*	*75%*

Bảng 3: Kết quả thống kê đánh giá thực nghiệm crawl ở chế độ standalone và Distributed – Trực quan hơn

(Lưu ý: Kết quả của quá trình thực nghiệm, tức các thời gian thực thi các giai đoạn được đo tính theo đơn vị giây. Tuy nhiên do khoảng thời gian quá dài, nên chúng tôi đã qui đổi ra thành phút, giờ và làm tròn một số phần lẻ không đáng kể ở những chỗ thích hợp.)

5.2.5 Đánh giá

Ta thấy với các giai đoạn mà khối lượng dữ liệu cần xử lý ít thì thời gian thực hiện bằng stand alone lại nhanh hơn thực hiện trên môi trường phân tán sử dụng MapReduce.

Khi dữ liệu cần xử lý lớn dần lên như khi các xử lý trong các vòng lặp crawl ở depth=2, depth=3 thì tốc độ xử lý trên hệ thống phân tán dần dần chiếm ưu thế so với xử lý cục bộ. Điển hình như khi depth=2 thì tổng thời thực hiện vòng lặp crawl của hệ thống phân tán bằng 88% của hệ thống cục bộ. Khi depth=3, lượng URL cần xử lý lên đến hơn 10000 thì tốc độ khi thực hiện phân tán bằng 69% khi thực hiện cục bộ. Điều này phù hợp với lý thuyết: MapReduce và HDFS sẽ thích hợp hơn với việc xử lý và lưu trữ các khối dữ liệu lớn.

Tổng thời gian thực hiện crawl trên hệ thống phân tán bằng 75% thời gian thực hiện crawl trên một máy đơn đã cho thấy được lợi ích của việc áp dụng tính toán phân tán vào Search Engine (quá trình crawl và index).

5.2.6 Kết luận

Kết quả còn chưa như mong đợi vì lượng dữ liệu xử lý còn nhỏ. Nếu ta thực hiện crawl sâu hơn (tăng depth lên) thì các kết quả có lẽ sẽ khả quan hơn. Tuy nhiên, việc thực hiện crawl sâu hơn đã gặp thất bại vì nguyên nhân sau: Khối lượng URL cần xử lý với depth=4 tăng lên khá cao. Thời gian thực hiện kéo dài ra khoảng vài

ngày. Mà khi hệ thống chạy được khoảng một hay hai ngày thì một số máy lại bị tình trạng reset (nguyên nhân có lẽ do điện áp hay lỏng dây cắm diện).

5.3 Thực nghiệm tìm kiếm trên tập chỉ mục

5.3.1 Mẫu dữ liệu:

Dữ liệu được có được từ quá trình crawl 10 trang web báo lớn ở Việt Nam với chiều sâu 4.

Số lượng trang web được nạp và index: 104000 trang web.

Kích thước khối dữ liệu: 2.5 GB

Thời gian thực hiện crawl (fetch + parse + index): 3 ngày.

5.3.2 Phần cứng

Phần cứng thực hiện thực nghiệm gồm:

- 1 máy (core2due, 2GB DDR2, 160GB HDD): is-teacher02
- 2 máy (Pentium 4, 512MB RDRam, 40GB HDD): is-aupelf04, is-teacher06

5.3.3 Phương pháp thực hiện

5.3.3.1 Tìm trên local (stand alone mode):

Dữ liệu được đặt toàn bộ trên hệ thống file cục bộ của máy is-aupelf04.

5.3.3.2 Tìm trên HDFS:

Dữ liệu được đặt trên hệ thống file phân tán với cấu hình

Namenode: is-aupelf04

Datanodes: is-teacher02, is-teacher06

5.3.3.3 Bổ dữ liệu ra và phân tán ra các Search servers

Mẫu dữ liệu được bổ thành hai phần đều nhau, không giao nhau (tức 2 mẫu con không có chung một URL nào) và phân phối lên 2 search server is-teacher02, isteacher06, port 2010.

5.3.4 Bảng kết quả thực hiện các truy vấn

Query	Thời gian thực thi			Tỷ lệ so sánh với Local		Số lượng kết
	HDFS	Local	Search server	HDFS	Search server	
"con người"	3762	398	205	945 %	52	6503
"bóng đá"	5003	443	411	1129 %	93	35258
"âm nhạc"	1329	211	194	630 %	92	16346

"thể thao"	3137	306	304	1025 %	99	51650
"xã hội"	1184	205	200	578 %	98	13922
"tác giả"	977	233	165	428 %	71	6431
"chuyên đề"	1524	168	181	907 %	108	1908
"gia đình"	1536	237	272	648 %	115	18944
"hệ thống thông tin"	8138	462	391	1761 %	85	127
"tổ chức"	4053	189	193	2144 %	102	16649
"tai nạn giao thông"	5669	221	212	2565 %	96	1663
"tình yêu" + "gia đình"	4672	301	309	1552 %	103	7087
"an ninh trật tự"	1495	197	260	759 %	132	115
"đời sống"	1211	155	162	781 %	105	5261
"nấu ăn"	429	81	69	530 %	85	1584
"văn hóa"	1246	163	161	764 %	99	13167
"địa điểm du lịch"	4003	456	312	878 %	68	41
"luật lệ"	958	165	130	581 %	79	209
"hình sự"	5038	313	268	1865 %	86	15149
"công an"	1959	317	182	618 %	57	3656
"an toàn giao thông"	3915	188	141	2082 %	75	223
"vệ sinh thực phẩm"	3129	327	411	957 %	126	130
"công ty"	1493	184	131	811 %	71	30591
"cá nhân"	1309	226	173	579 %	77	7112
"giải trí"	1970	227	185	868 %	81	22327
"trẻ em"	1627	198	163	822 %	82	6071
"giáo dục"	4124	190	96	2171 %	51	23190
"thị trường chuyển nhượng"	2523	177	153	1425 %	86	1045
"hình ảnh"	2715	200	164	1358 %	82	1045
"ngôi sao"	1510	233	163	648 %	70	19515
"thi đại học"	6442	341	219	1889 %	64	1997
"tuyển sinh"	1440	128	105	1125 %	82	8747
"thiị trường chứng khoán"	2553	138	135	1850 %	98	722
"game online"	726	184	186	395 %	101	3328

Bảng 4: Bảng thực hiện kết quả truy vấn

5.3.5 Đánh giá:

Qua kết quả trên, ta thấy việc tìm kiếm trên tập chỉ mục đặt trên HDFS là hoàn toàn không phù hợp, thời gian thực thi quá lâu (vượt hơn thời gian thực thi trên local nhiều lần), đúng như lý thuyết.

Đa số các câu truy phấn khi thực hiện phân tán đều cho kết quả tốt hơn khi thực hiện tập trung trên một máy, một số câu truy vấn cho tốc độ gần gấp đôi. Tuy nhiên do tập dữ liệu chưa đủ lớn, nên các kết quả này còn chưa thật sự thuyết phục.

Kết luận: Việc phân bổ tập chỉ mục và tìm kiếm trên các Search server đã mang lại kết quả là tốc độ tìm kiếm có tăng lên so với khi thực hiện trên một máy.

5.4. Kết luận, ứng dụng và hướng phát triển

5.4.1 Kết quả đạt được

Sau sự cố gắng trong mấy tháng vừa qua, luận văn đã đạt được một số kết quả sau đây:

 Tìm hiểu được nền tảng kiến thức về kiến trúc, chu kỳ, các thành phần ứng dụng và các mô hình cho dữ liệu lớn.

 Nghiên cứu cơ bản về kiểm soát truy xuất dữ liệu, đặc biệt là truy xuất cho dữ liệu lớn.

 Nghiên cứu về kiến trúc, các đặc điểm và cơ chế hoạt động của hai thành phần chính của Hadoop: HDFS và MapReduce Engine.

 Biết cách phát triển và triển khai ứng dụng theo mô hình MapReduce với Hadoop.

5.4.2 Ứng dụng

- Chứng khoáng Kis triển khai giải pháp ổ đĩa lưu trữ dữ liệu tầm trung của IBM để tăng cường khả năng lưu trữ và xử lý dữ liệu.
- Ngân hàng ACB xây dựng trung tâm dữ liệu dạng modun, ứng dụng các giải pháp phân tích kinh doanh của IBM nhằm xử lý các khối dữ liệu lớn.
- Hiện nay, Intel đang hỗ trợ cho thành phố Đà Nẵng triển khai các giải pháp liên quan đến dữ liệu lớn như biến trung tâm dữ liệu Đà Nẵng thành trung tâm dữ liệu xanh với công nghệ điện toán đám mây, tiến hành triển khai các phương án thử nghiệm (POC – Proof of concept), trong đó Intel sẽ chủ trì các POC về quản lý nguồn, trung tâm dữ liệu Intel sẽ tiếp tục hỗ trợ Đà Nẵng thiết lập 1 trung tâm dữ liệu theo chuẩn mở, nối kết mọi hệ thống dữ liệu trên địa bàn, phục vụ quản lý nhà nước và doanh nghiệp, phát triển các dịch vụ công trên nền công nghệ mạng hiện đại để cung cấp đến công dân và tổ chức.

5.4.3 Hướng phát triển

Điều khiển truy cập là một trong các biện pháp quan trọng nhằm đảm bảo an ninh, an toàn cho thông tin, hệ thống và mạng. Điều khiển truy cập thuộc lớp các biện pháp ngăn chặn tấn công, đột nhập. Luận văn nghiên cứu về dữ liệu lớn đồng thời các kỹ thuật điều khiển truy cập, bao gồm điều khiển truy cập tùy quyền (DAC), điều khiển truy cập bắt buộc (MAC), điều khiển truy cập dựa trên vai trò (RBAC) và điều khiển truy cập dựa trên luật (Rule-based AC). Cụ thể, các đóng góp của luận văn bao gồm:

- Nghiên cứu về kiến trúc, mô hình cho dữ liệu lớn
- Nghiên cứu tổng quan về điều khiển truy cập, các nguy cơ, điểm yếu và một số ứng dụng tiêu biểu của điều khiển truy cập.
- Nghiên cứu sâu về các kỹ thuật các kỹ thuật điều khiển truy cập, bao gồm điều khiển truy cập tùy quyền (DAC), điều khiển truy cập bắt buộc (MAC), điều khiển truy cập dựa trên vai trò (RBAC) và điều khiển truy cập dựa trên luật (Rule-based AC).
- Phân tích các kỹ thuật điều khiển truy cập được cài đặt trong các họ hệ điều hành phổ biến là Microsoft Windows và Unix/Linux.
- Đưa ra các khuyến nghị để đảm bảo an ninh, an toàn cho tài khoản, mật khẩu, thông tin và hệ thống.
- Đưa ra ứng dụng minh họa kiểm soát truy xuất dữ liệu theo mô hình MapReduce trên Framework Hadoop.

Luận văn có thể được nghiên cứu phát triển theo hướng sau:

- Nghiên cứu các giải pháp đảm bảo an ninh, an toàn hiệu quả cho các ứng dụng dựa trên điều khiển truy cập. Các cơ chế đảm bảo an toàn trong nhiều ứng dụng phổ biến như các ứng dụng trong kế toán, tài chính hiện đã có nhưng còn khá đơn giản, như chủ yếu dựa trên mật khẩu, không thực sự đảm bảo an toàn. Cần nghiên cứu phát triển các giải pháp đảm bảo an ninh, an toàn hiệu quả hơn cho các ứng dụng.
- Nghiên cứu các biện pháp điều khiển truy cập cho các hệ thống phân tán với các mục đích khác nhau.

TÀI LIỆU THAM KHẢO

[1] Wittenauer,A.(2008), *Deploying Grid Services Using Hadoop*, ApacheCon EU.

[2] Bertino, E., Bonatti, P.A., Ferrari (2001), *TRBAC: A temporal role -based access control model*, ACM TISSEC, 4(3), 191 -233.

[3] Bughin, J., Chui, M., & Manyika (2010), *Clouds, big data, and smart assets: Ten tech-enabled business trends to watch*. McKinsey Quarterly, 56(1), 75-86.

[4] Bertino, E., Ghinita, G., Kamra(2011), *Access Control for Databases: Concepts and Systems.* Now Publishers.

[5] http://blog.SQLAuthority.com

[6] Di Vimercati, S.De Capitani, Sara Foresti, and Pierangela Samarati (2008), *Recent advances in access control*, Handbook of Database Security, Springer US, pp. 1-26.

[7] Doug Cutting (2004), *Free Search: Lucene & Nutch*, Wizards of OS, Berlin.

[8] Doug Cutting (2005), *MapReduce in Nutch*, Yahoo!, Sunnyvale, CA, USA.

[9] Doug Cutting (2004), *Nutch:Open-Source Web Search Software*, University of Pisa, Italy.

[10] Doug Cutting (2004), *Nutch: Open Source Web Search*, New York.

[11] Kaisler, S., Armour, F., Espinosa, J. A.,Money (2013), *Big Data: Issues and Challenges Moving Forward.* HICSS 2013, pp. 995-1004.

[12] K..T. Smith (2014), "*Big Data Security: The Evolution of Hadoop's Security Model"*, InfoQ.

[13] Manyika, J., McKinsey Global Institute, Chui, M., Brown, B., Bughin, J., Dobbs, R.,Byers(2011),*Bigdata:Thenextfrontierforinnovation,competition,and productivity*, McKinsey Global Institute.

[14] Rajan, S. Etal (2012), *TopTen Big Data Security and Privacy Challenges*.

[15] Russom, 2011, *Big data analytics. TDWI Best Practices Report*, Fourth Quarter.

[16] Zikopoulos,P., & Eaton, 2011, *Understanding big data:Analytics for enterprise class hadoop and streaming data*, McGraw-Hill Osborne Media.

[17] W. Zeng, Y, Yang, B. Luo, 2013, *"Access Control for Big Data using Data Content*" in Proc,

[18] *"Big data to turn'mega' as capacity will hot 44 zettabytes by 2020*", DataIQ News32,http://www.dataiq.co.uk/news/20140410/big-data-turnmega-apacity-will-hit-44-zettabytes-2020.

[19] "*NoSQL Databases Explained*", mongoDBInc, http://www.mongodb.com/ nosql-explained, 2014.

[20] WhitePaper,Zittaset,http://www.zettaset.com/wpcontent/uploads/2014/04/ zettaset_wp_security_0413.pdf (2014). *"The Big Data Security Gap: Protecting the Hadoop Cluster*,"

[21] H.Mir,*"Hadoop Tutorial What is Hadoop*", http://ZeroTOProTraining.com, http://nusmv.irst.itc.it, ZeroToProTraining.

[22] Hadoop wiki: *http://wiki.apache.org/hadoop/.*

[23] Hadoop.apache.org

[24] http://www.baomoi.com/Thoi-dai-Big-Data-K1-Big-Data-la-gi/4308468.epi

[25] http://redis.io/topics/security, 2013, *Redis Security*.

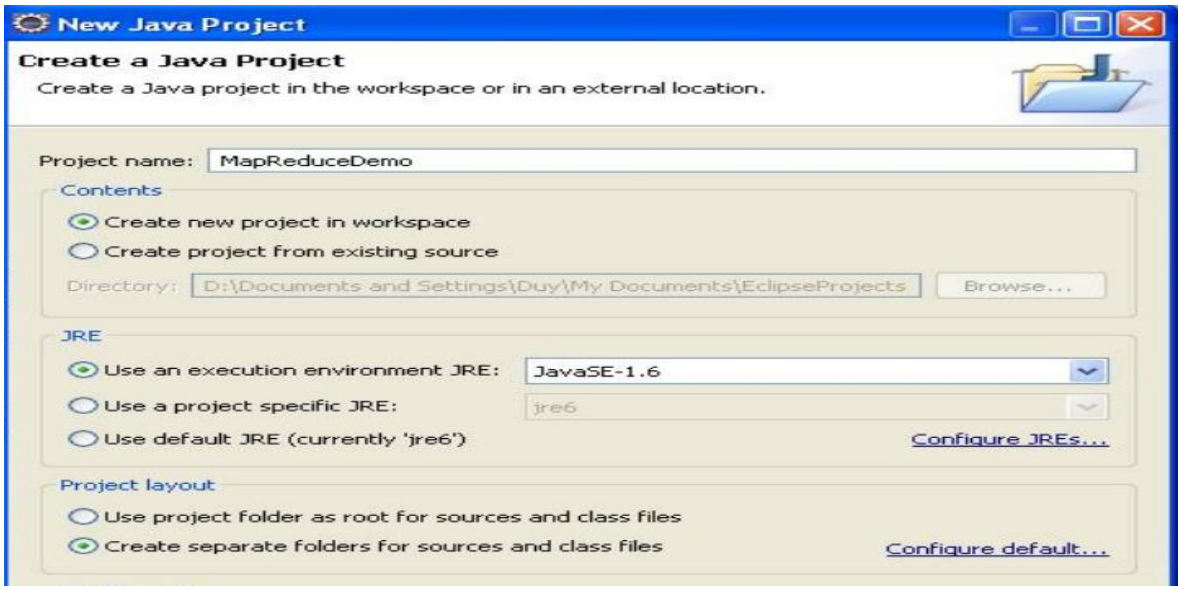
[26] http://hadoop.apache.org/docs/stable,2013,*ApacheHadoop Documentation*.

Phụ lục : Phát triển ứng dụng kiểm soát truy xuất dữ liệu theo mô hình MapReduce trên Framework Hadoop.

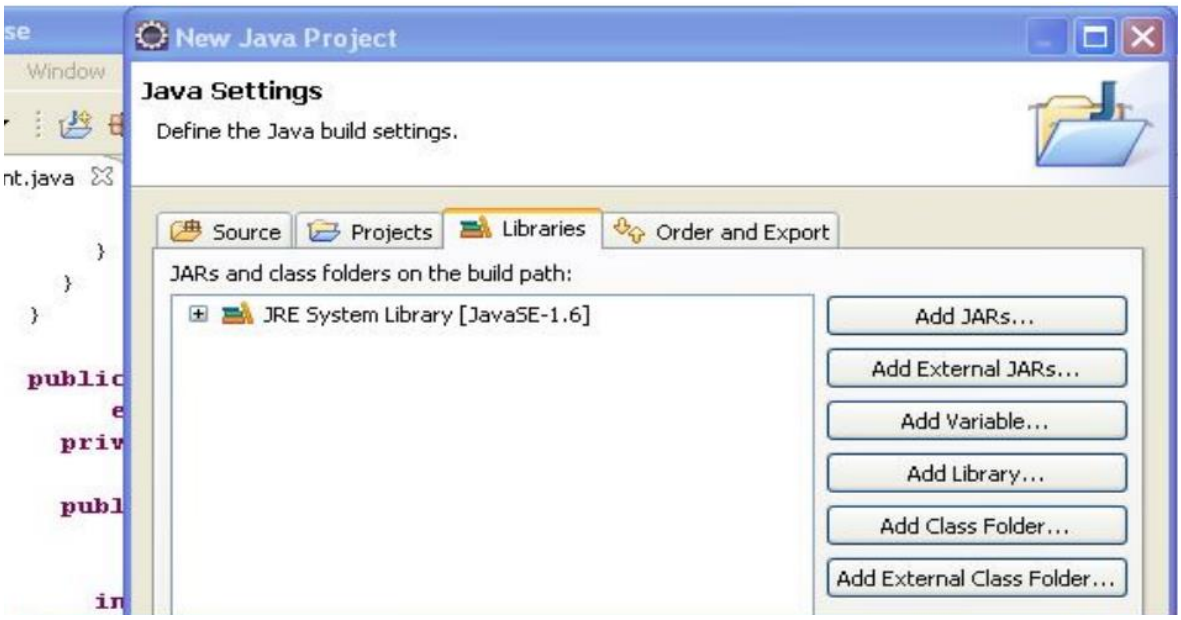
Trong chương này, trình bày một ứng dụng minh họa phổ biến nhất là ứng dụng đếm số lần xuất hiện của mỗi từ trong một file văn bản.

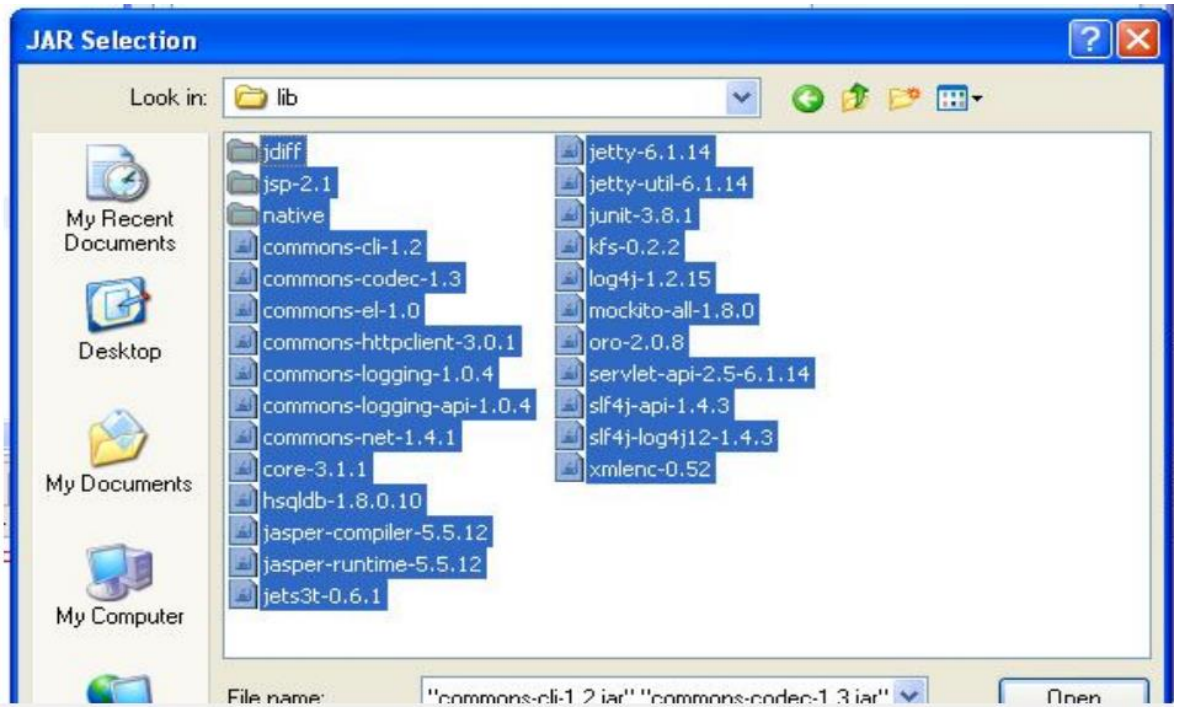
Bước đầu tiên, ta tạo project java (ở đây chúng tôi đặt tên là Jars…). Ta thực hiện 2 thao tác add lib sau:

 Đầu tiên ta add các lib thuộc thư mục cài đặt hadoop.

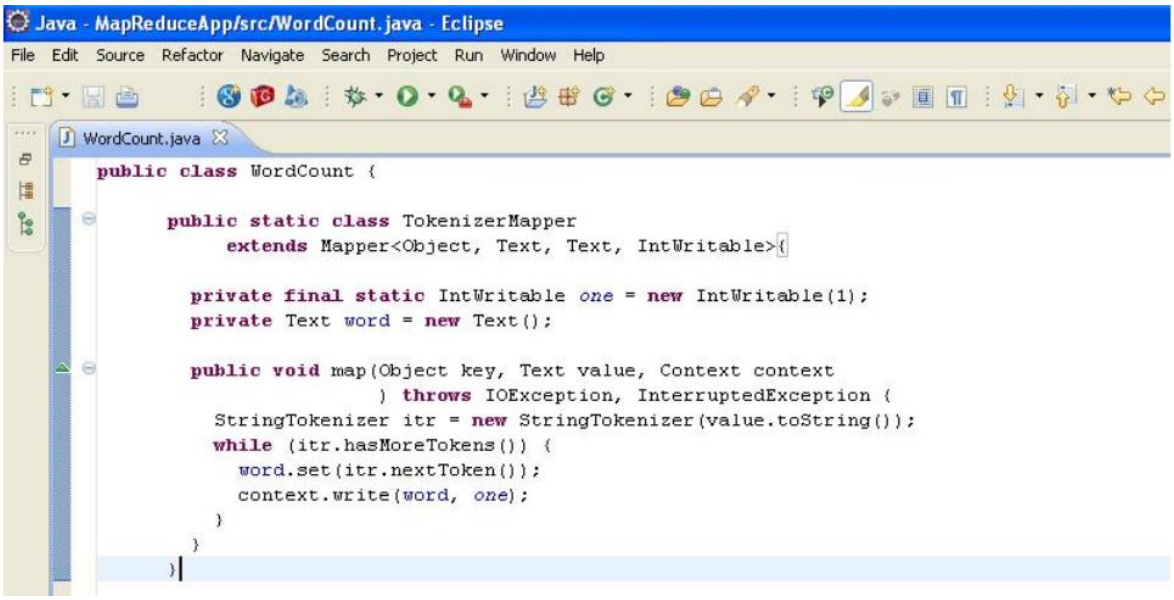


Sau đó, tiến hành add các lib từ thư mục <thư mục cài đặt hadoop>/lib.





Sau khi add thành công 2 loại lib trên, ta đã có đủ các api để tiến hành triển khai ứng dụng Hadoop MapReduce. Việc làm đầu tiên ta cần quan tâm là viết một lớp để định nghĩa hàm map. Lớp này phải extends lớp Mapper và bên trong phải định nghĩa cho phương thức map (phương thức này là hàm map trong mô hình MapReduce)



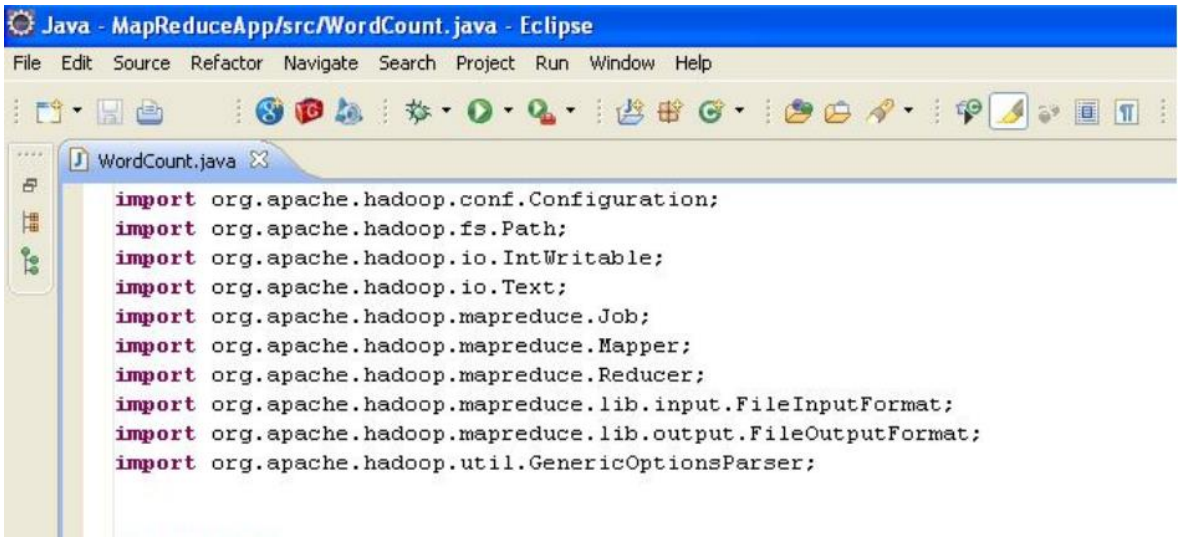
```
  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
```

Writable | Smart Insert | 32 : 8

Tiếp theo, ta viết một lớp để định nghĩa hàm reduce. Lớp này phải extends lớp Reducer và định nghĩa phương thức reduce (phương thức này được xem là hàm reduce trong mô hình MapReduce)



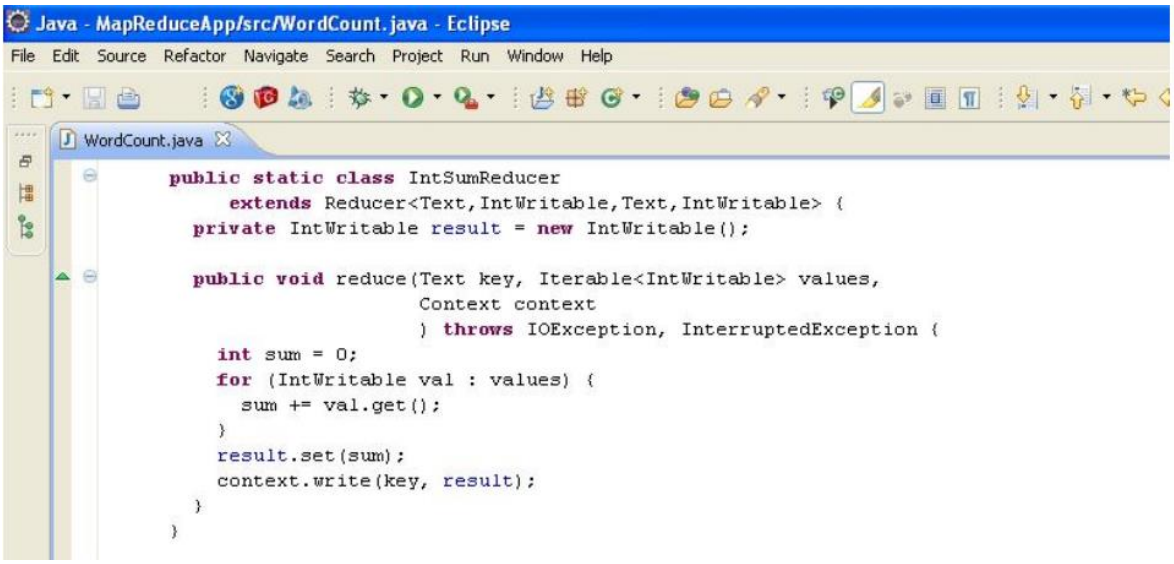
```
public class WordCount {

      public static class TokenizerMapper
           extends Mapper<Object, Text, Text, IntWritable>{

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context
                        ) throws IOException, InterruptedException {
          StringTokenizer itr = new StringTokenizer(value.toString());
          while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);
          }
        }
      }
```

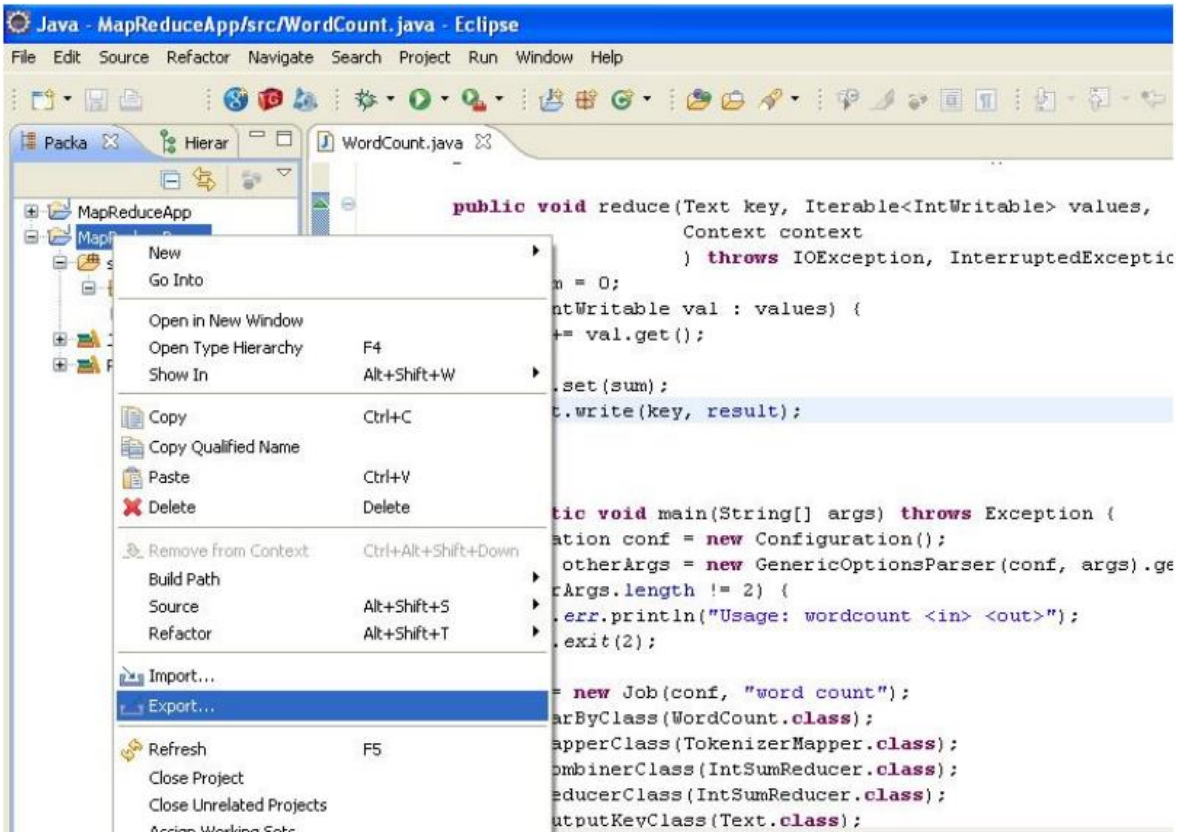
Sau khi có được 2 lớp định nghĩa cho hàm map và reduce. Ta tiến hành viết một lớp chính để thực hiện thao tác đệ trình công việc vào cho MapReduce Engine (Ở đây là JobTracker). Nhiệm vụ của ta trong việc viết lớp này khá đơn giản. Đầu tiên ta định nghĩa một đối tượng Configuration để lưu trữ các thông số cấu hình cũng như thông số để đệ trình công việc. Sau đó ta thiết lập từng thông số cho đối tượng Configuration như lớp thực hiện hàm map, lớp thực hiện hàm reduce, lớp thực hiện hàm combine, kiểu format cho key và n value của output cuối cùng cũng như kiểu format của file input và file output cuối cùng.



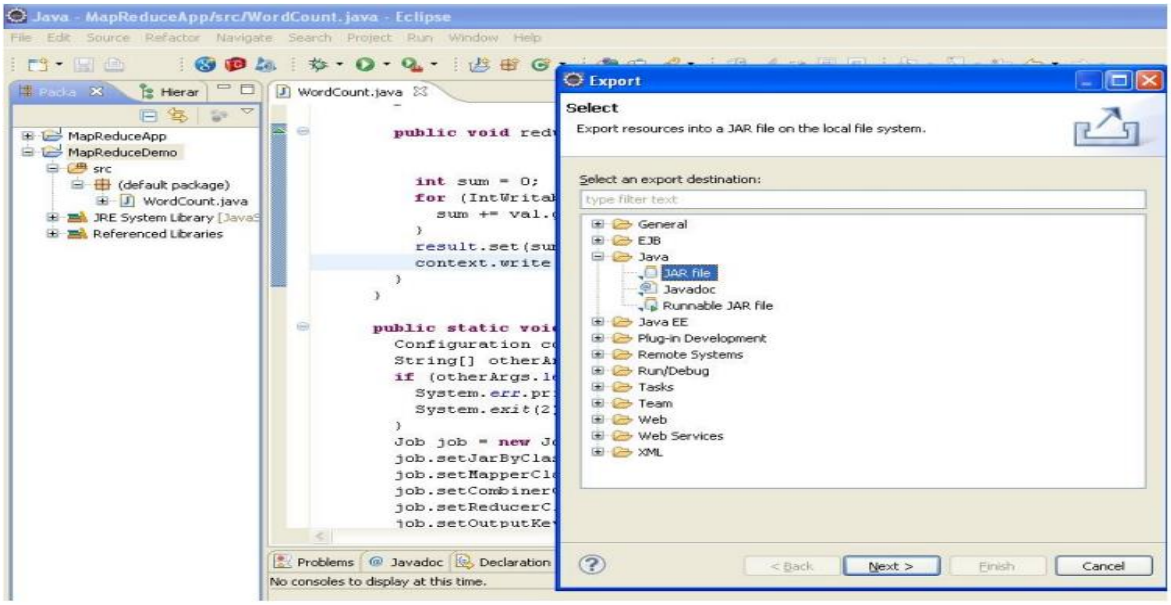
```
    public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
      if (otherArgs.length != 2) {
        System.err.println("Usage: wordcount <in> <out>");
        System.exit(2);
      }
      Job job = new Job(conf, "word count");
      job.setJarByClass(WordCount.class);
      job.setMapperClass(TokenizerMapper.class);
      job.setCombinerClass(IntSumReducer.class);
      job.setReducerClass(IntSumReducer.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
      FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
      FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
      System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Hadoop cung cấp một command để thực hiện chạy một ứng dụng Hadoop Mapreduce thông qua việc chạy file jar và hàm main của nó. Do đó, chúng ta sẽ export project thành file jar. Giả sử tên file jar là wordcount.jar. Dưới đây là các bước tạo file jar.

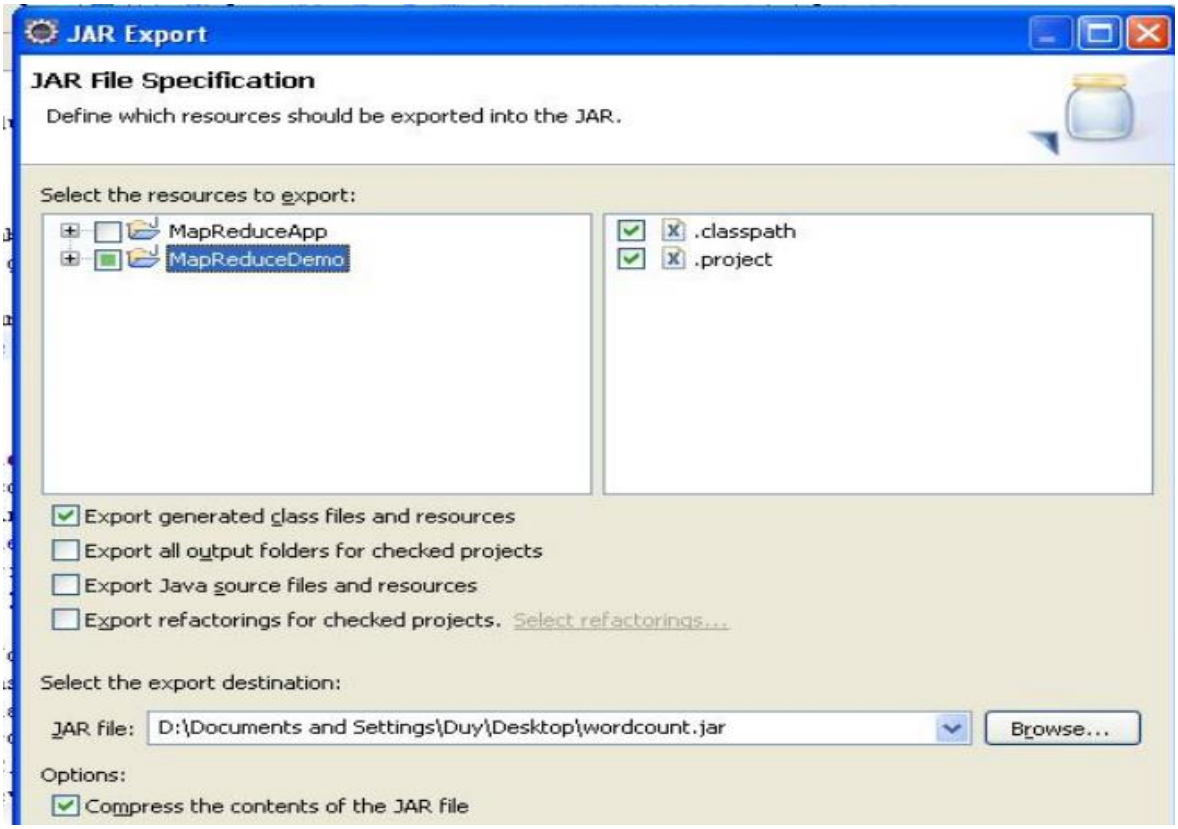
Đầu tiên right-click vào project chọn export



Sau đó chọn loại export là file jar.



Tiếp theo ta chọn project để export và chọn đường dẫn output cho file jar, rồi chọn Finish



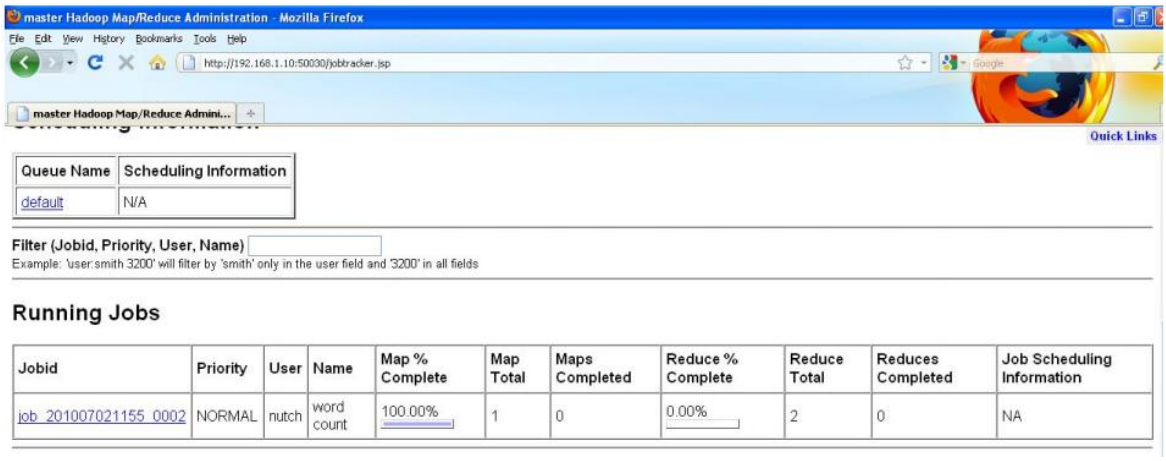
Sau khi có được file jar, ta thực hiện nó bằng command của hadoop như sau:

 hadoop jar wordcount.jar WordCount /inputtext.txt /output/

o WordCount: Lớp chứa hàm main để đệ trình job

o /inputtext.txt: File input nằm trên HDFS

o /output/ : Thư mục chứa các file output nằm trên HDFS (Số file output bằng với số reduce task). Vào trình quản lý của MapReduce ta xem tiến độ của thực thi job này. (Cluster ở đây gồm một master và 2 slave. Do dữ liệu inputtext.txt chỉ gồm vài KB nên chỉ có một map task thực thi)



Và kết quả đạt được sau khi thực hiện job wordcount thông qua trình quản lý HDFS. Do có 2 reduce task nên có 2 file output cuối cùng.