

**ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

Trần Văn Bách

**CƠ SỞ DỮ LIỆU PHÂN TÁN VÀ ỨNG DỤNG
TRONG MÁY TÌM KIẾM**

KHOÁ LUẬN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY

Ngành: Công nghệ thông tin

HÀ NỘI - 2010

**ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

Trần Văn Bách

**CƠ SỞ DỮ LIỆU PHÂN TÁN VÀ ỨNG DỤNG
TRONG MÁY TÌM KIẾM**

KHOÁ LUẬN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY

Ngành: Công nghệ thông tin

Cán bộ hướng dẫn: ThS. Nguyễn Thu Trang

HÀ NỘI - 2010

Lời cảm ơn

Trước tiên tôi xin gửi lời cảm ơn và lòng biết ơn sâu sắc nhất tới Thạc sĩ Nguyễn Thu Trang, người đã tận tình chỉ bảo và hướng dẫn tôi trong suốt quá trình thực hiện khóa luận tốt nghiệp này.

Tiếp theo, tôi xin cảm ơn các thầy cô, Ban giám hiệu nhà trường đã tạo cho tôi những điều kiện tốt nhất để tôi có thể học tập và nghiên cứu tại trường Đại học Công Nghệ.

Tôi cũng xin gửi lời cảm ơn chân thành nhất đến chị Nguyễn Hoàng Quỳnh cũng như các thầy cô, các anh chị và các bạn sinh viên tại phòng thí nghiệm SIS đã giúp đỡ nhiệt tình và tạo điều kiện cho tôi hoàn thành phần thực nghiệm của khóa luận này.

Cuối cùng, tôi muốn gửi lời cảm ơn vô hạn tới gia đình, bạn bè và những người thân yêu đã luôn động viên, cổ vũ tôi trong suốt quá trình thực hiện khóa luận tốt nghiệp.

Một lần nữa, tôi xin chân thành cảm ơn !

Tóm tắt

Đi đôi với sự phát triển ngày càng nhanh của khoa học, kỹ thuật đó là sự phát triển của công nghệ cơ sở dữ liệu. Các hệ cơ sở dữ liệu truyền thống, quản lý dữ liệu theo phương thức tập trung đôi khi đã không còn phù hợp với các hệ thống hiện đại. Hệ phân tán, tối ưu hơn đã ngày càng được sử dụng rộng rãi và phổ biến.

Khóa luận tốt nghiệp với đề tài “*Hệ cơ sở dữ liệu phân tán và ứng dụng trong máy tìm kiếm*” tập trung tìm hiểu về kiến trúc, cách thức hoạt động của hệ thống lưu trữ lớn Bigtable, hệ thống quản lý dữ liệu phân tán Hadoop. Khóa luận cũng tiến hành cài đặt thử nghiệm hệ thống Hadoop lưu trữ phân tán với cụm máy tính để bàn kết nối trên mạng LAN ứng dụng cho máy tìm kiếm mã nguồn mở Nutch.

Mục lục:

Tóm tắt	- 2 -
Danh sách các hình	- 6 -
Chương 1: Giới thiệu hệ cơ sở dữ liệu phân tán	- 7 -
1.1. Nhu cầu về hệ phân tán.....	- 7 -
1.2. Định nghĩa hệ CSDL phân tán.....	- 7 -
1.3. Ưu điểm của hệ CSDL phân tán	- 8 -
1.4. Nhược điểm của hệ CSDL phân tán	- 9 -
Chương 2: Dữ liệu máy tìm kiếm và cơ sở dữ liệu Bigtable	- 10 -
2.1. Giới thiệu về Bigtable và dữ liệu máy tìm kiếm.....	- 10 -
2.2. Mô hình dữ liệu.....	- 11 -
2.2.1. Hàng	- 11 -
2.2.2. Họ cột.....	- 12 -
2.2.3. Nhãn thời gian.....	- 13 -
2.3. Giao diện lập trình ứng dụng API.....	- 13 -
2.4. Xây dựng các khối	- 15 -
2.5. Thực thi.....	- 15 -
2.5.1. Định vị bảng phụ.....	- 16 -
2.5.2. Chỉ định bảng phụ.....	- 18 -
2.5.3. Phục vụ bảng phụ.....	- 19 -
2.5.4. Nén.....	- 20 -
2.6. Lọc	- 21 -
2.7. Ước lượng hiệu năng	- 25 -
Chương 3: Hệ thống quản lý file phân tán Hadoop	- 28 -

3.1.	Khái niệm cơ bản về hệ thống Hadoop	- 28 -
3.1.1.	Kiến trúc của Hadoop	- 28 -
3.1.2.	Job Tracker và Task Tracker: các máy MapReduce	- 30 -
3.2.	Cơ chế MapReduce	- 32 -
3.2.1.	Giới thiệu	- 32 -
3.2.2.	Các thành phần logic.....	- 33 -
3.2.2.1.	Map	- 33 -
3.2.2.2.	Reduce.....	- 33 -
3.2.3.	Mô hình luồng dữ liệu.....	- 35 -
3.2.4.	Đánh giá	- 37 -
3.3.	Ứng dụng của Hadoop	- 38 -
3.3.1.	Hadoop trong máy tìm kiếm Yahoo	- 38 -
3.3.2.	Hadoop trên các dịch vụ Amazon EC2/S3	- 38 -
3.3.3.	Hadoop với Sun Grid Engine.....	- 39 -
Chương 4:	Kiến trúc HBase	- 40 -
4.1.	Giới thiệu HBase.....	- 40 -
4.2.	Mô hình dữ liệu.....	- 40 -
4.2.1.	Khung nhìn khái niệm.....	- 40 -
4.2.2.	Khung nhìn lưu trữ vật lý.....	- 41 -
4.3.	Kiến trúc và thực thi	- 43 -
4.3.1.	HBaseMaster.....	- 43 -
4.3.2.	HRegionServer.....	- 44 -
4.3.3.	HBase Client	- 46 -
Chương 5:	Cài đặt thực nghiệm và đánh giá hiệu năng.....	- 47 -
5.1.	Môi trường thử nghiệm.....	- 47 -

5.2.	Cài đặt cụm Hadoop phân tán quy mô 3 máy.....	- 47 -
5.3.	Chạy thử và đánh giá hiệu năng.....	- 52 -
	Kết luận.....	- 55 -
	Tài liệu tham khảo	- 56 -

Danh sách các hình

Hình 1: Thứ tự lưu trữ một trang web

Hình 2: Thứ bậc định vị bảng phụ

Hình 3: Số lần đọc và ghi trên 1 giây với 1000 byte dữ liệu.

Hình 4: Kiến trúc tổng thể của Hadoop

Hình 5: Các máy Map Reduce

Hình 6: Thành phần logic Mapper và Reducer

Hình 7: Sơ đồ luồng dữ liệu

Hình 8: Cấu hình file hadoop-site.xml

Hình 9: Giao diện namenode

Hình 10: Giao diện JobTracker

Hình 11: Kết quả chạy ví dụ WordCount

Hình 12: Kết quả file output

Chương 1: Giới thiệu hệ cơ sở dữ liệu phân tán

1.1. Nhu cầu về hệ phân tán

Công nghệ cơ sở dữ liệu (CSDL) đã trải qua một quá trình hình thành và phát triển khá lâu dài. Ban đầu, các hệ CSDL thường gắn liền với ứng dụng, nghĩa là mỗi ứng dụng định nghĩa và duy trì dữ liệu của riêng chúng. Phát triển hơn, dữ liệu được quản lý một cách tập trung, nhiều ứng dụng khác nhau có thể truy vấn vào CSDL tập trung đó. Việc xây dựng những hệ CSDL tập trung này có nhiều lợi ích, một lợi ích điển hình đó là tính độc lập dữ liệu. Độc lập dữ liệu được hiểu là nếu chúng ta có thay đổi về tổ chức logic hay tổ chức vật lý của dữ liệu thì cũng không ảnh hưởng gì đến các ứng dụng sử dụng dữ liệu đó và ngược lại. Tuy nhiên, CSDL tập trung cũng tồn tại nhiều khuyết điểm, có thể kể đến đó là khi trung tâm dữ liệu có sự cố thì toàn hệ thống sẽ ngừng hoạt động, hay tình trạng tắc nghẽn khi có quá nhiều yêu cầu truy xuất vào CSDL.

Hệ CSDL phân tán ra đời đã phần nào khắc phục được những điểm yếu của CSDL tập trung. Là kết quả của sự hợp nhất của hai hướng tiếp cận đối với quá trình xử lý dữ liệu: công nghệ CSDL và công nghệ mạng máy tính. CSDL phân tán gồm nhiều CSDL tích hợp lại với nhau thông qua mạng máy tính để trao đổi dữ liệu, thông tin. CSDL được tổ chức và lưu trữ ở những vị trí khác nhau trong mạng máy tính và chương trình ứng dụng làm việc trên cơ sở truy cập dữ liệu ở những điểm khác nhau đó.

Có thể thấy nguyên lý phân tán cũng tương tự như nguyên lý “chia để trị” đã phổ biến rất rộng rãi. Một bài toán lớn và phức tạp được chia thành nhiều bài toán nhỏ và đơn giản hơn, giao cho nhiều đơn vị thực hiện sau đó tổng hợp kết quả lại. Xét trên khía cạnh người dùng, đặc biệt là các công ty, xí nghiệp, thì việc xử lý phân tán đáp ứng tốt hơn với việc phân bố ngày càng rộng rãi của các tổ chức này.

1.2. Định nghĩa hệ CSDL phân tán.

M. Tamer Ozsu và Patrick Valduriez[1] định nghĩa một CSDL phân tán là “một tập hợp nhiều CSDL có liên đới logic và được phân bố trên một mạng máy tính”. Từ đó hai tác giả đã định nghĩa hệ quản trị CSDL phân tán là một hệ thống phần mềm cho phép quản lý các hệ CSDL phân tán và làm cho các hệ phân tán trở nên “vô hình” đối với người sử dụng.

Hai điểm quan trọng được nêu ra trong định nghĩa là:

- Phân bố trên một mạng máy tính: Dữ liệu không cư trú trên một vị trí. Điều này giúp phân biệt một CSDL phân tán với một CSDL tập trung, đơn lẻ.
- Liên đới logic: Dữ liệu có một số các thuộc tính ràng buộc chúng với nhau, điều này giúp chúng ta phân biệt một CSDL phân tán với một tập hợp CSDL cục bộ hoặc các tệp cư trú tại các vị trí khác nhau trong một mạng máy tính.

1.3. Ưu điểm của hệ CSDL phân tán

- Về tổ chức và tính kinh tế: Ngày càng xuất hiện nhiều tổ chức với quy mô lớn, các chi nhánh của những tổ chức này phân bố ở nhiều nơi có vị trí địa lý rất xa nhau. Việc sử dụng một hệ tập trung với những tổ chức như này là không hợp lý, phân tán là giải pháp phù hợp. Cùng với sự phát triển của kinh tế thương mại hiện nay, các trung tâm máy tính tập trung cũng không còn phù hợp, việc phân tán trở thành nhu cầu cần thiết.
- Tận dụng, liên kết những CSDL sẵn có: có thể tạo nên một CSDL phân tán từ những CSDL cục bộ đã có sẵn. Tiến trình này có thể yêu cầu phải sửa đổi các CSDL cục bộ.
- Thuận lợi cho việc mở rộng: Các tổ chức có thể mở rộng, thêm vào các đơn vị mới một cách dễ dàng, đơn vị mới vừa có tính tự trị vừa có kết nối với tổ chức. Với CSDL tập trung, cũng có thể ước lượng khởi tạo một kích thước lớn để mở rộng về sau, tuy nhiên việc này là rất khó khăn, nếu khởi tạo quá lớn mà không dùng hết thì lãng phí tài nguyên, khởi tạo kích thước nhỏ thì có thể không đủ dùng.
- Giảm chi phí truyền thông: Trong hệ phân tán, một chương trình ứng dụng tại địa phương có thể giảm bớt được chi phí truyền thông nếu sử dụng bản sao dữ liệu có tại địa phương.
- Cải thiện hiệu suất: Hệ CSDL phân tán có thể tăng số lượng công việc thực hiện qua áp dụng nguyên lý xử lý song song với hệ thống xử lý đa nhiệm. Hệ CSDL phân tán cũng có lợi trong việc phân tán dữ liệu, tạo ra các chương trình ứng dụng chạy tại nhiều máy trong mạng. Các nơi xử lý có thể hỗ trợ lẫn nhau, xung đột giữa các bộ vi xử lý là tối thiểu. Tải được chia sẻ giữa các bộ vi xử lý, do đó giảm được hiện tượng tắc nghẽn do thắt cổ chai trong mạng.

- Tính tin cậy và sẵn sàng: Độ tin cậy và tính sẵn sàng là một trong những mục đích của hệ CSDL phân tán. Tuy nhiên để đạt được điều này không dễ dàng. Khả năng tự trị tại các vị trí khác nhau khiến cho tính tin cậy cao của toàn bộ hệ thống khó được đảm bảo. Sự cố trong hệ phân tán có thể thường xuyên xảy ra hơn trong hệ tập trung, do cấu trúc thành phần phức tạp hơn, nhưng hậu quả của sự cố chỉ giới hạn ở mức cục bộ, sự sụp đổ của toàn bộ hệ thống là rất hiếm khi xảy ra.

1.4. Nhược điểm của hệ CSDL phân tán

Tuy có những ưu điểm vượt trội so với CSDL tập trung, CSDL phân tán có những điểm yếu cần cân nhắc khi sử dụng mà có thể tóm gọn lại trong 4 vấn đề sau:

- Tính phức tạp: Hệ phân tán phức tạp hơn hệ tập trung, ngoài các vấn đề cần giải quyết như tập trung, còn có các vấn đề khác như về mạng hay về đồng bộ hóa.
- Chi phí: một hệ phân tán đòi hỏi phải có thêm các thiết bị phần cứng mới (thiết bị truyền thông....), các phần mềm và phương pháp truyền thông phức tạp hơn, và đặc biệt là chi phí về nhân lực. Vì thế cần phải phân tích cẩn thận giữa những lợi ích mà nó mang lại với chi phí để thiết kế, sử dụng và bảo trì nó.
- Phân tán quyền điều khiển: điều khiển phân tán là một trong những ưu điểm của hệ CSDL phân tán. Tuy nhiên sự phân tán phải đi kèm với quá trình đồng bộ hóa. Việc điều khiển phân tán có thể trở thành một gánh nặng nếu không có những chiến lược phù hợp để giải quyết chúng.
- Tính an ninh (bảo mật): Trong CSDL tập trung, người quản trị có thể kiểm soát được các truy xuất dữ liệu. An ninh dễ dàng được kiểm soát ở trung tâm. Tuy nhiên đối với hệ phân tán, các máy được kết nối qua mạng máy tính, việc đảm bảo an ninh trong môi trường mạng là phức tạp hơn.

Chương 2: Dữ liệu máy tìm kiếm và cơ sở dữ liệu Bigtable

2.1. Giới thiệu về Bigtable và dữ liệu máy tìm kiếm

Bigtable[11] là một hệ thống lưu trữ phân tán dùng để quản lý dữ liệu có cấu trúc được thiết kế để co giãn trong phạm vi rất lớn: hàng petabyte dữ liệu thông qua hàng nghìn server. Nhiều dự án tại Google lưu trữ dữ liệu bằng Bigtable, có thể kể đến chỉ mục Web, Google Earth, và Google Finance. Những ứng dụng này đặt ra những yêu cầu khác nhau đối với Bigtable, xét cả trong phạm vi của kích thước dữ liệu (từ URL tới trang web tới các hình ảnh vệ tinh) và các yêu cầu về độ trễ (từ những xử lý chính đến việc phục vụ dữ liệu thời gian thực). Mặc dù những yêu cầu này rất khác nhau, Bigtable đã cung cấp thành công một giải pháp linh động, hiệu năng cao cho tất cả các sản phẩm của Google. Chương này mô tả mô hình dữ liệu được cung cấp bởi Bigtable, và thiết kế thực thi của Bigtable, cho phép người dùng điều khiển kiến trúc và định dạng dữ liệu Bigtable.

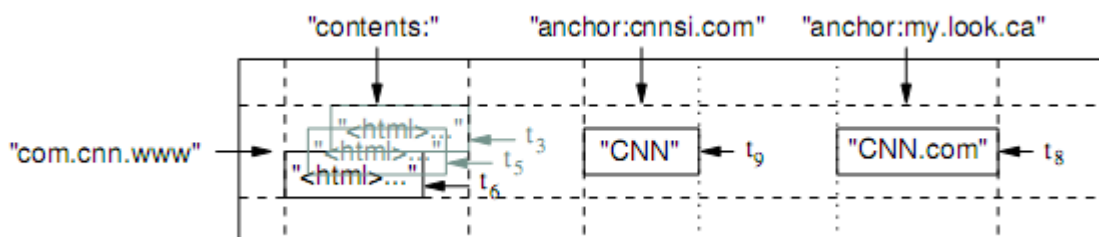
Bigtable được thiết kế, thực thi và phát triển trong vòng 2 năm rưỡi. Bigtable đã đạt được: tính ứng dụng lớn, tính co giãn, hiệu năng cao và tính sẵn sàng cao. Bigtable đã được sử dụng trong hơn 60 dự án và sản phẩm của Google, bao gồm Google Analytic, Google Finance, Orkut, Tìm kiếm cá nhân, Writely, ... Những sản phẩm này sử dụng Bigtable khác nhau, sắp xếp từ các công việc xử lý theo khối hướng thông lượng tới việc phục vụ dữ liệu với độ trễ thấp tới người dùng cuối. Những cụm Bigtable được sử dụng với nhóm hàng nghìn server, và lưu trữ tới vài trăm terabyte dữ liệu. Bigtable tương tự như một cơ sở dữ liệu, và nó chia sẻ nhiều sự quản lý thực thi với CSDL. CSDL song song [9] và CSDL tập trung [10] đều có khả năng co giãn và hiệu năng cao, nhưng Bigtable cung cấp giao diện cho mỗi hệ thống khác nhau. Bigtable không hỗ trợ mô hình dữ liệu quan hệ đầy đủ. Thay vào đó, nó cung cấp các ứng dụng client với một mô hình dữ liệu đơn giản có hỗ trợ điều khiển động đối với kiến trúc và định dạng dữ liệu. Bigtable cho phép các ứng dụng client suy ra những đặc tính vị trí của dữ liệu được mô tả trong kho lưu trữ bên dưới. Dữ liệu được đánh chỉ mục theo tên hàng và cột có thể là các xâu bất kì. Bigtable cũng coi dữ liệu như là các xâu không diễn dịch (uninterpreted), mặc dù các ứng dụng client thường sắp xếp những dạng khác nhau của dữ liệu có cấu trúc và bán cấu trúc vào những xâu này. Client có thể điều khiển vị trí của dữ liệu của họ thông

qua những lựa chọn cẩn thận ở lược đồ. Cuối cùng, những lược đồ tham số Bigtable cho phép client kiểm soát phục vụ dữ liệu trong hoặc ngoài đĩa.

2.2. Mô hình dữ liệu

Một Bigtable là một bản đồ phân tán, đa chiều ổn định. Bản đồ này được đánh chỉ mục bởi một khóa hàng, khóa cột, và một nhãn thời gian, mỗi giá trị trong bản đồ là một mảng dữ liệu không diễn dịch (uninterpreted):

$(\text{row: string}, \text{column: string}, \text{time: int64}) \rightarrow \text{string}$



Hình 1: Ví dụ về lưu trữ một trang web

Ví dụ về lưu trữ trang “cnn.com”: Tên hàng là địa chỉ URL, họ cột “contents:” chứa nội dung trang, họ cột “anchor” chứa văn bản của bất kì liên kết nào tới trang web. Trang cnn được 2 trang tham chiếu tới, do đó hàng chứa các cột có tên là anchor:cnnsi.com và anchor:my.look.ca. Mỗi ô anchor có nhiều phiên bản, cột “contents:” có 3 phiên bản với nhãn thời gian là t3, t5, t6.

Giả sử rằng chúng ta muốn giữ một bản sao của một tập hợp lớn các trang web và thông tin liên quan mà có thể được sử dụng bởi nhiều dự án khác nhau; chúng ta gọi những bảng này là Webtable. Trong Webtable, chúng ta sử dụng địa chỉ URL như là các khóa hàng, các bộ phận khác nhau của trang web như là tên cột, và lưu trữ nội dung trang Web vào CONTENTS, và cột dưới nhãn thời gian khi chúng được lấy ra.

2.2.1. Hàng

Các khóa hàng là các chuỗi bất kì (dung lượng có thể lên tới 64KB, mặc dù hầu hết người dùng chỉ sử dụng 10-100B). Tất cả các hoạt động đọc hay ghi dữ liệu bên dưới một khóa hàng đơn đều là “nguyên tử” (không quan tâm đến số cột được đọc và được ghi

trong hàng), một giải pháp thiết kế có thể làm cho các ứng dụng khách thấy dễ dàng hơn khi suy luận về nguyên lý của hệ thống khi xảy ra cập nhật đồng thời lên cùng một hàng.

Bigtable bảo trì dữ liệu theo thứ tự từ điển bởi khóa hàng. Dãy các hàng được phân cách động. Mỗi một dãy hàng được gọi là bảng phụ (tablet), bảng phụ là đơn vị của phân tán và cân bằng tải. Việc đọc các dãy hàng ngắn có hiệu quả và yêu cầu giao tiếp với chỉ một số lượng nhỏ các máy. Client có thể khai thác thuộc tính này bằng cách chọn những khóa hàng của họ vì thế họ có được những vị trí tốt cho việc truy cập dữ liệu. Ví dụ, trong Webtable, các trang trong cùng tên miền được nhóm vào các hàng kề nhau bằng cách đảo ngược các thành phần trong địa chỉ URL. Ví dụ, chúng ta lưu dữ liệu cho địa chỉ `maps.google.com/index.html` bằng khóa `com.google.maps/index.html`. Lưu trữ các trang có tên miền giống nhau gần nhau giúp cho các host và phân tích tên miền được hiệu quả hơn.

2.2.2. Họ cột

Các khóa cột được nhóm vào một bảng được gọi là “họ” cột, tạo thành các khối cơ bản của kiểm soát truy xuất. Tất cả dữ liệu được lưu trong một “họ” cột thường có chung kiểu (do chúng ta nén dữ liệu trong cùng một họ đồng thời với nhau). Một “họ” cột phải được tạo ra trước khi dữ liệu được lưu trữ tại một cột nào đó trong họ. Sau khi một họ được tạo, mọi khóa cột bên trong họ đó đều có thể sử dụng. Số họ cột trong một bảng không nhiều (nhiều nhất là hàng trăm), và những họ này hiếm khi thay đổi trong quá trình hoạt động. Ngược lại, một bảng có số cột không giới hạn.

Một khóa cột được đặt tên dựa theo cú pháp ‘tên_họ:tính_chất’. Ví dụ về họ cột cho Webtable là LANGUAGE, nó lưu trữ ngôn ngữ mà trang web đó được viết. Chúng ta chỉ sử dụng một khóa cột cho họ LANGUAGE, và nó lưu trữ định danh của ngôn ngữ của mỗi trang web. Một họ cột cũng rất hữu dụng cho bảng này là ANCHOR; mỗi cột trong họ đại diện cho một anchor đơn lẻ. Phần tính chất là tên của trang liên quan, nội dung ô là kết nối văn bản.

Điều khiển truy xuất cùng với đĩa và tính toán bộ nhớ được thực hiện tại mức họ cột. Trong ví dụ Webtable, bộ điều khiển cho phép chúng ta quản lý vài loại ứng dụng khác nhau: một vài trong số chúng dùng để tạo mới dữ liệu cơ bản, một vài để đọc dữ liệu cơ bản và tạo ra các họ cột từ đó, và một vài thì chỉ cho phép xem dữ liệu đang tồn tại.

2.2.3. Nhân thời gian

Mỗi ô trong Bigtable có thể chứa nhiều phiên bản của cùng một dữ liệu, những phiên bản này được đánh chỉ mục bởi nhãn thời gian. Nhãn thời gian là các số nguyên 64 bit. Chúng có thể được chỉ định bởi Bigtable, trong trường hợp chúng mô tả thời gian thực tới từng micro giây, hoặc được chỉ định bởi các ứng dụng người dùng. Ứng dụng nào cần tránh các xung đột phải tự sinh ra nhãn thời gian duy nhất của riêng chúng. Các phiên bản khác của một ô được lưu trữ theo thứ tự giảm dần của nhãn thời gian, nhờ đó phiên bản mới nhất có thể được đọc trước.

Để cho việc quản lý các phiên bản dữ liệu được dễ dàng hơn, cho phép hỗ trợ hai môi trường trên các họ cột. Phía client có thể chỉ định một số n nào đó phiên bản cuối cùng được giữ lại, hoặc chỉ giữ lại những phiên bản đủ mới (ví dụ, chỉ giữ lại giá trị được ghi trong vòng 7 ngày trở lại).

Trong ví dụ Webtable, chúng ta đặt các nhãn thời gian cho các trang đã được duyệt lưu trữ trong CONTENT:, chính là thời gian mà trang được duyệt. Cơ chế lọc rác (garbage-collect) cho phép chúng ta chỉ giữ lại 3 phiên bản mới nhất của mọi trang web.

2.3. Giao diện lập trình ứng dụng API

Bigtable API cung cấp chức năng cho việc tạo và xóa các bảng và các họ cột. Nó cũng cung cấp chức năng để chuyển cụm (cluster), bảng, và siêu dữ liệu họ cột.

Các ứng dụng khách có thể ghi và xóa giá trị, tìm kiếm giá trị từ các hàng riêng lẻ, hoặc lặp lại 1 nhóm dữ liệu trong một bảng. Dưới đây là một đoạn code C++ sử dụng hàm RowMutation để thực hiện một chuỗi cập nhật. Gọi hàm Apply thực hiện một sự thay đổi nguyên tử đến Webtable: thêm 1 anchor vào www.cnn.com và xóa 1 anchor khác đi.

```
//Open a table
Table *T = OpenorDie("/bigtable/web/webtable");

//Write a new anchor and delete an old anchor
RowMutation r1 (T, "com.cnn.www");

r1.Set("anchor:www.c-span.org", "CNN");

r1.Delete("anchor:www.abc.com");
```


Operation op;

Apply(&op, &r1);

Đoạn code dưới đây cho thấy đoạn code C++ sử dụng hàm Scanner để lặp lại tất cả các anchor trong 1 hàng. Client có thể lặp lại trên nhiều họ cột, và có vài cơ chế định ra giới hạn số hàng, cột, nhãn thời gian tạo ra bởi 1 bộ scan. Ví dụ, chúng ta có thể hạn chế bộ scan chỉ tạo ra những anchor có cột phù hợp với biểu thức anchor.*.cnn.com, hoặc chỉ tạo ra những anchor mà nhãn thời gian trong vòng 10 ngày trở lại.

```
Scanner scanner(T);
```

```
ScanStream *stream;
```

```
stream= scanner.FetchColumnFamily("anchor");
```

```
stream-> SetReturnAllVersions();
```

```
scanner.Lookup("com.cnn.www");
```

```
for (; !stream->Done(); stream->next()) {
```

```
printf ("%s %s %11d %s \n", scanner.Rowname(), stream->Columnname(), stream->MicroTimestamp(), stream->Value());
```

```
}
```

Bigtable hỗ trợ một vài tính năng khác cho phép người dùng vận dụng dữ liệu theo nhiều cách phức tạp:

- Hỗ trợ giao tác đơn hàng (single-row transaction), có thể sử dụng để thực hiện chuỗi đọc-sửa-ghi nguyên tử trên dữ liệu lưu trữ dưới một khóa hàng đơn.
- Hỗ trợ sự thực thi của những kịch bản client-supplied trong không gian địa chỉ của server. Kịch bản được viết bằng ngôn ngữ phát triển tại Google dành cho việc xử lý dữ liệu gọi là Sawzall [15]. Hiện tại, API dựa trên nền Sawzall không cho phép các kịch bản client viết lại vào Bigtable, nhưng nó cho phép các dạng khác nhau của phép biến đổi dữ liệu, bộ lọc dựa trên biểu thức bất kì.

Bigtable có thể sử dụng với MapReduce [8], một framework dùng để chạy các tính toán song song phát triển bởi Google.

2.4. Xây dựng các khối

Bigtable được xây dựng trên các phần khác nhau của cơ sở hạ tầng của Google. Bigtable sử dụng hệ thống file phân tán Google (distributed Google File System) [13] để lưu trữ bản ghi và file dữ liệu. Một cụm Bigtable hoạt động trong một nhóm các máy được chia sẻ, các máy này chạy nhiều ứng dụng phân tán khác nhau, và các tiến trình Bigtable thường chia sẻ máy tính với tiến trình từ các ứng dụng khác. Bigtable phụ thuộc vào hệ thống quản lý cụm trong việc lên lịch công việc, quản lý tài nguyên khi chia sẻ, giải quyết sự cố, và kiểm tra trạng thái của máy. Định dạng file Google SSTable được sử dụng để lưu trữ dữ liệu Bigtable. Một SSTable cung cấp một bản đồ liên tục, và thứ tự ko đổi từ các khóa tới các giá trị, nơi mà cả khóa và giá trị đều là các xâu bất kỳ. Các phép toán được cung cấp để tìm kiếm giá trị liên quan đến khóa được chỉ rõ, và để lặp lại tất cả các cặp khóa/giá trị trong một dải khóa được chỉ ra. Sâu hơn nữa, mỗi Sstable mang một chuỗi các block (mỗi block có kích thước 64KB, có thể điều chỉnh được). Một chỉ mục block (lưu tại cuối của Sstable) được sử dụng để định vị block; chỉ mục được tải vào bộ nhớ khi SStable được mở.

Bigtable dựa vào một dịch vụ khóa phân tán có tính sẵn sàng cao gọi là Chubby [5]. Một dịch vụ Chubby bao gồm 5 mô hình hoạt động, một trong số chúng được chọn làm chủ và đáp ứng các yêu cầu. Dịch vụ này chỉ “sống” khi phần lớn các mô hình đang chạy và có giao tiếp với các mô hình khác. Chubby sử dụng thuật toán Paxos [6] để giữ các mô hình của nó nhất quán trong trường hợp có lỗi xảy ra. Chubby cung cấp một không gian tên bao gồm các thư mục và các file nhỏ. Mỗi thư mục hoặc file có thể sử dụng như 1 khóa, việc đọc và ghi file là tự động. Thư viện clien của Chubby cung cấp một nơi lưu trữ nhất quán cho file Chubby. Mỗi client Chubby duy trì một phiên với một dịch vụ Chubby.

Bigtable sử dụng Chubby để: bảo đảm chỉ có duy nhất một mô hình chủ tại mọi thời điểm; để lưu trữ vị trí khởi động của dữ liệu Bigtable để lưu trữ thông tin lược đồ Bigtable (thông tin về họ cột cho mỗi bảng), và để lưu trữ danh sách điều khiển truy xuất.

2.5. Thực thi

Thực thi Bigtable có 3 thành phần chính: một thư viện được kết nối tới mọi client, một máy chủ, và nhiều máy chủ phụ. Máy chủ phụ có thể được thêm hoặc gỡ bỏ động từ một cụm để điều tiết những thay đổi của tải làm việc.

Máy chủ chính có trách nhiệm chỉ định các bảng phụ (tablet) vào các máy chủ phụ, phát hiện sự bổ sung cũng như mở rộng của máy chủ phụ, cân bằng tải, và loại bỏ file trong GFS. Thêm vào đó, nó điều khiển những thay đổi lược đồ ví dụ như việc tạo ra các bảng hay các họ cột.

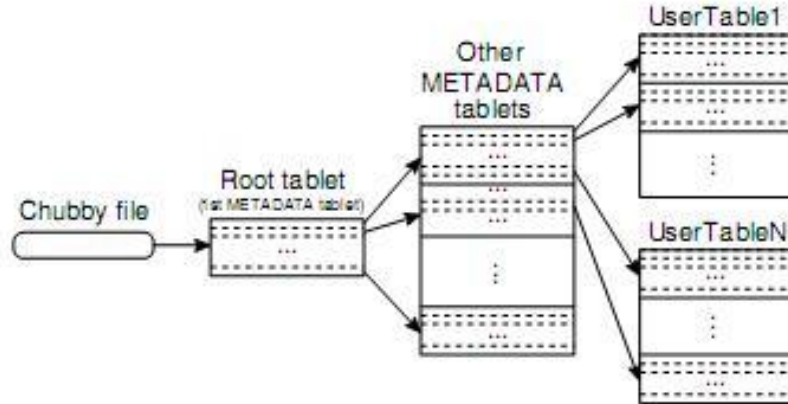
Mỗi máy chủ phụ quản lý một tập các bảng phụ (thông thường có từ khoảng 10 đến 1000 bảng phụ trên một máy chủ phụ). Máy chủ phụ quản lý các yêu cầu đọc và ghi vào các bảng con đã được tải, và chia nhỏ các bảng khi chúng quá lớn.

Như với các hệ thống lưu trữ phân tán một máy chủ [13], dữ liệu khách không được đưa qua máy chủ, client giao tiếp trực tiếp với các máy chủ phụ để đọc và ghi. Bởi client Bigtable không phụ thuộc vào máy chủ về thông tin vị trí các bảng phụ, hầu hết client không bao giờ giao tiếp với máy chủ. Do đó, máy chủ không phải chịu tải lớn.

Một cụm Bigtable lưu trữ một số bảng. Mỗi bảng gồm có một tập các bảng phụ, và mỗi bảng phụ mang toàn bộ dữ liệu kết hợp với một dải các hàng. Khởi đầu mỗi bảng chỉ gồm một bảng phụ và khi phát triển, nó tự động chia thành nhiều bảng phụ, với kích thước tiêu chuẩn trong khoảng 100-200M.

2.5.1. Định vị bảng phụ

Chúng ta sử dụng hệ thứ bậc 3 cấp tương tự như cây B+ [7] để lưu trữ thông tin định vị bảng phụ hình 2.



Hình 2: Thứ bậc định vị bảng phụ

Cấp thứ nhất là một file được lưu trữ tại Chubby chứa vị trí của bảng phụ gốc. Bảng phụ gốc chứa vị trí của tất cả các bảng phụ trong một bảng Metadata đặc biệt. Mỗi bảng Metadata chứa vị trí của một tập các bảng phụ người dùng. Bảng phụ gốc là bảng đầu tiên trong bảng Metadata, nhưng được đối xử đặc biệt, không bao giờ bị chia cắt, để đảm bảo rằng thứ bậc không bao giờ quá 3 cấp.

Bảng Metadata lưu trữ vị trí của một bảng phụ bên dưới khóa hàng là một mã hóa của định danh của tên bảng chứa bảng phụ đó và hàng kết thúc của nó. Mỗi hàng Metadata lưu trữ xấp xỉ 1KB dữ liệu trong bộ nhớ. Với giới hạn 128MB, lược đồ định vị 3 mức đủ đánh địa chỉ 2^{34} bảng phụ (hoặc 2^{61} byte trong 128M bảng phụ).

Thư viện client lưu trữ vị trí bảng phụ. Nếu client không biết về vị trí của bảng phụ, hoặc nếu nó phát hiện ra vị trí lưu trữ là sai, nó sẽ di chuyển đệ quy lên theo thứ bậc. Nếu bộ đệm của client là rỗng, thuật toán định vị yêu cầu ba lần quay vòng trong mạng, bao gồm một lần đọc từ Chubby. Nếu bộ đệm client cũ, thuật toán định vị có thể lên tới 6 vòng. Mặc dù vị trí bảng phụ được lưu trong bộ nhớ, vì thế nếu không có yêu cầu GFS, chúng ta giảm giá thành trong hầu hết trường hợp bằng cách có một thư viện nạp trước vị trí các bảng phụ: nó đọc dữ liệu metadata của nhiều hơn một bảng phụ bất cứ khi nào nó đọc bảng Metadata.

Chúng ta cũng lưu trữ thông tin thứ hai trong bảng Metadata, bao gồm một bản ghi tất cả các sự kiện liên quan đến bảng phụ (ví dụ như khi một máy chủ bắt đầu hoạt động). Thông tin này hữu ích cho việc debug và phân tích hiệu năng.

2.5.2. Chỉ định bảng phụ

Mỗi bảng phụ được phân vào một máy chủ phụ vào một thời điểm. Máy chủ chính lưu vết các thiết lập của máy chủ phụ đang hoạt động, và sự phân công hiện tại của các bảng phụ tới các máy chủ, bao gồm bảng phụ nào chưa được chỉ định. Khi một bảng phụ ko được chỉ định, và một máy chủ phụ có đủ khả năng cho bảng phụ sẵn sàng, máy chủ chính sẽ phân công bảng phụ bằng cách gửi một yêu cầu tải bảng phụ tới máy chủ phụ.

Bigtable sử dụng Chubby để lưu vết các máy chủ phụ. Khi một máy chủ phụ khởi động, nó tạo ra, và yêu cầu một khóa dành riêng, một file với tên duy nhất trong thư mục riêng Chubby. Máy chủ chính giám sát thư mục này (gọi là server directory) để phát hiện ra các máy chủ phụ. Một máy chủ phụ ngừng phục vụ nếu nó mất khóa của nó: ví dụ, do việc phân chia mạng làm cho máy chủ mất phiên làm việc Chubby của nó. (Chubby cung cấp một cơ chế hiệu quả cho phép một máy chủ phụ nó có còn giữ khóa của nó mà ko bị ảnh hưởng bởi tắc nghẽn mạng). Một máy chủ phụ sẽ cố gắng giành lại một khóa dành riêng trên file của nó chỉ cần file đó còn tồn tại. Nếu file ko còn tồn tại, máy chủ phụ không bao giờ có thể phục vụ trở lại, vì thế nó tự ngừng hoạt động. Bất cứ khi nào một máy chủ phụ ngừng hoạt động (ví dụ, do hệ thống quản lý cụm gỡ bỏ máy chủ ra khỏi cụm) nó cố gắng giải phóng khóa của nó nhờ đó máy chủ chính có thể chỉ định lại những bảng phụ này nhanh chóng hơn.

Máy chủ chính có trách nhiệm phát hiện khi một máy chủ phụ không còn phục vụ các bảng phụ của nó, và phân công lại bảng phụ sớm nhất có thể. Để phát hiện khi một máy chủ phụ ngừng phục vụ, máy chủ chính hỏi một cách định kì mỗi máy chủ phụ trạng thái khóa của nó. Nếu một máy chủ phụ báo cáo rằng nó đã mất khóa, hoặc nếu máy chủ chính ko thể kết nối tới máy chủ phụ trong lần thử cuối cùng của nó, máy chủ chính sẽ cố gắng giành lại khóa trên file của máy chủ phụ. Nếu máy chủ chính có thể giành được khóa, Chubby sẽ hoạt động và máy chủ phụ sẽ ngừng hoạt động hoặc gặp vấn đề khi kết nối tới Chubby, vì thế máy chủ chính đảm bảo rằng máy chủ phụ không bao giờ phục vụ trở lại bằng cách xóa file của nó. Một khi file của máy chủ phụ đã bị xóa, máy chủ chính có thể di chuyển tất cả các bảng phụ trước đó đã được phân cho máy chủ phụ đó về tập các bảng phụ chưa được chỉ định. Để đảm bảo rằng cụm Bigtable không bị nguy hiểm bởi các vấn đề mạng giữa máy chủ chính và Chubby, máy chủ chính tự ngừng hoạt động

nếu phiên Chubby của nó hết thời gian. Tuy nhiên, như đã nói ở trên, máy chủ chính gặp sự cố không ảnh hưởng đến sự chỉ định các bảng phụ vào các máy chủ phụ.

Khi một máy chủ chính được khởi động bởi hệ thống quản lý cụm, nó cần phải phát hiện ra sự phân công bảng phụ hiện tại trước khi nó thay đổi chúng. Máy chủ chính thực hiện những bước sau: 1: Máy chủ chính chiếm lấy một khóa máy chủ chính duy nhất trên Chubby; 2: nó scan thư mục trong Chubby để tìm ra những máy chủ phụ đang hoạt động; 3: nó giao tiếp với tất cả các máy chủ phụ đang hoạt động để tìm ra những bảng phụ nào đã được chỉ định cho mỗi máy chủ phụ; 4: máy chủ chính scan bảng Metadata để học tập các bảng phụ. Bất cứ khi nào scan thấy một bảng phụ chưa được phân công rồi, nó bổ sung thêm các bảng phụ vào tập các bảng chưa được chỉ định, từ đó chọn ra bảng thích hợp để phân công.

Việc scan bảng Metadata không thể được thực hiện cho đến khi các bảng phụ Metadata được phân công. Bởi vậy, trước khi scan (bước 4) máy chủ chính bổ sung thêm các bảng phụ gốc vào tập các bảng chưa được phân công nếu phát hiện ra một phân công bảng phụ gốc trong bước 3. Sự bổ sung này đảm bảo bảng phụ gốc sẽ được phân công. Vì bảng phụ gốc chứa tất cả thông tin của tất cả các bảng phụ Metadata, nên máy chủ chính biết về tất cả chúng sau khi scan được bảng phụ gốc.

Tập các bảng phụ đang tồn tại chỉ thay đổi khi một bảng được tạo ra hay xóa đi, hai bảng phụ đang tồn tại được gộp thành một bảng phụ lớn hơn, hoặc một bảng phụ bị chia thành hai bảng phụ nhỏ hơn. Máy chủ chính có thể lưu vết những thay đổi này. Những bảng phụ bị chia cắt được đối xử đặc biệt khi chúng được khởi tạo bởi một máy chủ phụ. Máy chủ phụ thực thi việc tách bằng cách ghi lại thông tin cho bảng phụ mới trong bảng Metadata. Khi một hoạt động tách được chuyển giao, nó báo cho máy chủ chính. Trong trường hợp thông báo bị mất (do máy chủ chính hoặc phụ lỗi), máy chủ chính phát hiện ra bảng phụ mới bằng cách yêu cầu một máy chủ phụ tải bảng phụ bị tách. Máy chủ phụ báo lại cho máy chủ chính về việc chia tách.

2.5.3. Phục vụ bảng phụ

Trạng thái liên tục của bảng phụ được lưu tại GFS. Cập nhật được thực thi vào một bản ghi thực thi lưu trữ các bản ghi làm lại (redo). Những lần cập nhật này, những cập

nhật gần hơn được lưu trong bộ nhớ đệm được sắp xếp gọi là memtable , những cập nhật cũ hơn được lưu trữ theo trình tự trong Sstable.

Để phát hiện ra một bảng phụ, một máy chủ phụ đọc dữ liệu metadata của nó từ bảng Metadata . Dữ liệu metadata này chứa danh sách Sstable bao gồm một bảng phụ và một tập các điểm làm lại, chúng là những con trỏ trỏ vào bất kì bản ghi thực thi nào có thể chứa dữ liệu của bảng phụ. Máy chủ phụ đọc những chỉ số của SSTable vào bộ nhớ và tổ chức lại memtable bằng cách áp dụng tất cả những cập nhật được thực thi từ điểm làm lại.

Khi thực hiện ghi trên máy chủ phụ, máy chủ phụ kiểm tra rằng nó được định dạng tốt (well-formed), và người gửi được cho phép thực hiện sự thay đổi. Sự cho phép được thực hiện bằng cách đọc danh sách những người ghi đã được cho phép từ file Chubby. Một thay đổi hợp lệ được viết vào bản ghi thực thi. Nhóm thực thi được sử dụng để tăng thông lượng của nhiều thay đổi nhỏ [5, 12]. Sau khi ghi hoàn tất, nội dung của nó đã được chèn vào memtable.

Khi thực hiện đọc trên máy chủ phụ, nó cũng kiểm tra định dạng tốt và quyền hạn tương tự. Một hoạt động đọc hợp lệ được thực thi trên một khung nhìn hợp nhất của chuỗi của SStable và memtable. Từ khi SStable và memtable sắp xếp cấu trúc dữ liệu theo trình tự từ điển, khung nhìn hợp nhất được tạo thành hiệu quả hơn.

2.5.4. Nén

Do việc thực thi hoạt động ghi, kích thước của memtable tăng lên. Khi kích thước của memtable đạt đến giới hạn, memtable sẽ đóng băng, một memtable mới được tạo ra, và memtable bị đóng băng được chuyển vào SStable và ghi vào GFS. Bộ xử lý nén nhỏ này có 2 mục đích: nó rút ngắn bộ nhớ sử dụng bởi máy chủ phụ, và giảm lượng dữ liệu được đọc từ bản ghi thực thi trong quá trình hồi phục nếu máy chủ phụ bị lỗi. Hoạt động đọc và ghi sắp tới có thể tiếp diễn khi đang nén.

Mọi bộ nén nhỏ đều tạo ra một SStable mới. Nếu chế độ này không được kiểm tra liên tục, các hoạt động đọc có thể cần phải kết hợp với cập nhật từ một số bất kì của Sstable. Thay vào đó, chúng ta giới hạn số file bằng cách thực thi định kì việc nén gộp (merging compaction) trên nền . Nén gộp đọc nội dung của một vài Sstable và memtable,

và ghi ra một SStable mới. SSable và memtable đầu vào có thể được loại bỏ sớm nhất khi việc nén hoàn thành.

Nén gộp ghi lại tất cả Sstabe vào chính xác một SStable được gọi là nén lớn (major compaction). Sstable tạo ra bởi nén non-major có thể chứa những mục xóa đặc biệt, cấm việc xóa dữ liệu trong SStalbe cũ hơn khi chúng vẫn còn hoạt động. Một bộ nén lớn, mặt khác, tạo ra một SStable ko chứa thông tin xóa hay dữ liệu đã bị xóa.

Bigtable quay vòng qua tất cả bảng phụ của nó và áp dụng nén lớn một cách đều đặn lên chúng. Nén lớn cho phép Bigtable phục hồi tài nguyên sử dụng bởi dữ liệu đã bị xóa, và cũng cho phép nó để đảm bảo rằng những dữ liệu đã bị xóa biến mất khỏi hệ thống, điều này rất quan trọng để máy chủ lưu trữ những thông tin nhạy cảm.

2.6. Lọc

Sự thực thi được mô tả trong các chương trước yêu cầu một số lần lọc dữ liệu để có thể đạt được hiệu quả cao, tính sẵn sàng, và tính tin cậy cho người dùng. Chương này mô tả các phần của việc thực thi một cách chi tiết hơn nhằm làm nổi bật quá trình lọc.

Locality Groups (Các nhóm địa phương)

Client có thể nhóm vài họ cột vào thành một nhóm địa phương. Một SStable được sinh ra cho mỗi nhóm địa phương trong mỗi bảng phụ. Việc cô lập các họ cột mà không truy xuất điển hình được vào cùng một nhóm địa phương giúp cho việc đọc hiệu quả hơn. Ví dụ, trang metadata trên Wehtable có thể nhóm vào một nhóm, và nội dung của trang có thể vào một nhóm khác: một ứng dụng muốn đọc dữ liệu metadata không cần phải đọc qua tất cả các trang nội dung.

Thêm vào đó, một vài tham số điều chỉnh hữu ích có thể được chỉ rõ trên một nền tảng nhóm địa phương. Ví dụ, một nhóm địa phương có thể được trình bày trong bộ nhớ (in-memory). SStable cho các nhóm địa phương in-memory được tải một cách chậm chạp vào bộ nhớ của máy chủ phụ. Mỗi lần tải, những họ cột thuộc về nhóm địa phương có thể được đọc mà không cần truy xuất đĩa. Tính năng này rất hữu dụng cho những mảnh dữ liệu nhỏ được truy xuất thường xuyên: chúng ta sử dụng nó một cách nội tại bên trong cho việc định vị các họ cột ở bảng Metadata.

Nén

Client có thể điều khiển dù cho Sstable của các nhóm địa phương có được nén hay ko. Và nếu thế, thì định dạng nén nào được sử dụng. Định dạng nén user-specified được áp dụng cho mỗi block SStable (kích thước có thể kiểm soát được thông qua tham số biến đổi đặc trưng). Mặc dù chúng ta mất vài khoảng trống để nén mỗi block riêng biệt, chúng ta được lợi từ các phần của một Sstable có thể đọc mà không cần giải nén toàn bộ file. Nhiều client sử dụng một lược đồ nén tùy chỉnh hai giai đoạn. Giai đoạn thứ nhất sử dụng lược đồ Bentley và McIlroy [3], nén những xâu dài bằng ngang một cửa sổ rộng. Giai đoạn thứ hai sử dụng một thuật toán nén nhanh, tìm kiếm những bản sao trong một cửa sổ dữ liệu nhỏ cỡ 16KB. Cả hai quá trình nén đều rất nhanh, chúng mã hóa khoảng 100-200MB/s, và giải mã 400-1000MB/s ở những máy tính hiện đại.

Thậm chí mặc dù chúng ta nhấn mạnh tốc độ thay vì làm giảm dung lượng khi lựa chọn thuật toán nén, lược đồ nén hai giai đoạn vẫn làm tốt một cách đáng ngạc nhiên. Ví dụ, trong Webtable, chúng ta sử dụng lược đồ nén này để lưu trữ nội dung trang Web. Trong một thí nghiệm, chúng ta lưu trữ một lượng lớn tài liệu tại một nhóm địa phương. Nhằm mục đích thí nghiệm, chúng ta tự giới hạn chỉ có một phiên bản của mỗi tài liệu thay vì lưu trữ tất cả các phiên bản. Lược đồ đã giảm bớt được dung lượng xuống 10 lần. Nó tốt hơn nhiều so với nén Gzip điển hình, chỉ từ 3 đến 4 lần trên các trang HTML bởi vì cách mà các hàng Webtable được sắp xếp: tất cả các trang từ một host đơn lẻ được lưu trữ cùng nhau. Điều này cho phép thuật toán Bentley-McIlroy nhận dạng một lượng lớn các mẫu soạn sẵn được chia sẻ từ host tương tự. Nhiều ứng dụng, ko chỉ Webtable, chọn tên các hàng của chúng tương tự như dữ liệu kết thúc cụm, nhờ đó đạt được tỉ lệ nén tốt. Tỉ lệ nén thậm chí còn tốt hơn khi chúng ta lưu trữ nhiều phiên bản của cùng một giá trị tại Bigtable.

Bộ đệm và hiệu năng đọc

Để cải thiện hiệu năng đọc, các máy chủ phụ sử dụng bộ đệm hai mức. Mức cao hơn là Scan Cache, nó lưu trữ các cặp khóa/giá trị được gửi lại bởi giao diện SStable thành các mã máy chủ phụ. Block Cache là mức thấp hơn, lưu trữ các block SStable đã được đọc từ GFS. Scan Cache hiệu quả nhất với các ứng dụng có xu hướng đọc dữ liệu lặp lại nhiều lần. Block Cache hữu ích cho các ứng dụng có xu hướng đọc dữ liệu gần với

dữ liệu chúng vừa đọc (ví dụ đọc liên tiếp, hoặc đọc bất kì từ các cột khác nhau trong một nhóm địa phương).

Bộ lọc Bloom

Như mô tả ở trên, một hoạt động đọc phải đọc từ tất cả các SStable làm thành trạng thái của bảng phụ. Nếu những Sstable này không có trong bộ nhớ, chúng ta có thể kết thúc việc truy xuất đĩa quá nhiều. Chúng ta giảm số lần truy xuất bằng cách cho phép client chỉ định rõ là bộ lọc Bloom [4] sẽ được tạo ra cho SStable trong nhóm địa phương riêng biệt. Một bộ lọc Bloom cho phép chúng ta hỏi một Sstable có thể mang bất kì dữ liệu nào cho một cặp hàng/cột đã được chỉ định được hay không. Đối với một vài ứng dụng, một lượng nhỏ bộ nhớ của máy chủ phụ sử dụng để lưu trữ bộ lọc Bloom giảm số lần yêu cầu tìm kiếm trên đĩa một cách mạnh mẽ. Sử dụng bộ lọc bloom cũng đưa đến hệ quả là hầu hết việc tìm kiếm các hàng hoặc cột không tồn tại không cần thiết phải chạm vào đĩa.

Thi hành các bản ghi thực thi

Nếu chúng ta giữ những bản ghi thực thi cho mỗi bảng phụ trong một file bản ghi riêng biệt, một lượng lớn file sẽ được ghi đồng thời vào GFS. Những hoạt động ghi này có thể phải tìm kiếm trên đĩa rất nhiều lần để ghi vào những bản ghi file vật lý khác nhau. Thêm vào đó, có nhiều file bản ghi trên một bảng phụ cũng giảm hiệu quả của việc tối ưu thực thi nhóm, làm các nhóm có xu hướng nhỏ hơn. Để khắc phục hậu quả này, chúng ta ghép thêm những biến đổi vào một bản ghi thực thi đơn lẻ trên mỗi máy chủ phụ, trộn lẫn những biến đổi của những bảng khác nhau và một log file vật lý.

Sử dụng một bản ghi có những lợi ích đáng kể về hiệu năng trong những hoạt động thông thường, nhưng nó khó khôi phục. Khi một máy chủ phụ chết, các bảng phụ mà nó phục vụ sẽ được chuyển tới một số lượng lớn máy chủ phụ khác: mỗi máy chủ tải một số ít các bảng phụ của máy chủ bị chết. Để phục hồi trạng thái của các bảng phụ, máy chủ mới phải áp dụng lại những thay đổi đối với bảng đó từ bản ghi thực thi được ghi bởi máy chủ cũ. Tuy nhiên, những thay đổi của những bảng này đã được trộn lẫn trong những log file vật lý.

Chúng ta tránh việc trùng lặp những bản ghi bằng cách sắp xếp những bản ghi theo thứ tự (tên bảng, tên hàng, số dãy bản ghi). Tại đầu ra đã được sắp xếp, tất cả những thay đổi cho một bảng phụ cụ thể nào đó được đặt kề nhau và từ đó có thể được đọc hiệu quả chỉ với một lần tìm kiếm trên đĩa. Để song song hóa việc sắp xếp, chúng ta phân chia các log file thành các mảnh 64MB, và sắp mỗi mảnh song song vào các máy chủ phụ riêng biệt. Quá trình sắp xếp này được phối hợp bởi máy chủ chính và được khởi tạo khi một máy chủ phụ cho biết là nó cần phát hiện những thay đổi từ một vài file bản ghi thực thi. Việc ghi các bản ghi thực thi vào GFS đôi khi gây ra một vài trục trặc do một vài lý do nào đó (ví dụ, một máy chủ GFS có lỗi ghi, hoặc tắc nghẽn mạng, hoặc quá tải). Để bảo vệ những thay đổi từ các nhánh của GFS, mỗi máy chủ phụ trên thực tế có hai tuyến ghi, mỗi tuyến ghi vào file bản ghi của chính nó; chỉ một trong hai tuyến được hoạt động trong một thời điểm. Nếu việc ghi vào các file bản ghi hoạt động có hiệu suất kém, bộ ghi sẽ chuyển sang tuyến khác, và những thay đổi trong hàng đợi của bản ghi thực thi sẽ được ghi bởi tuyến ghi mới. Các mục bản ghi (log entry) chứa số thứ tự để cho phép quá trình khôi phục bỏ qua những mục trùng lặp do kết quả của việc chuyển tuyến ghi.

Tăng tốc khôi phục bảng phụ

Nếu máy chủ chính di chuyển một bảng phụ từ một máy chủ phụ này sang một máy chủ phụ khác, máy chủ phụ nguồn trước hết sẽ nén nhỏ dữ liệu tại bảng phụ đó. Việc nén này làm giảm thời gian khôi phục bằng cách giảm số trạng thái chưa nén chặt tại bản ghi thực thi của máy chủ phụ. Sau quá trình nén này, máy chủ phụ ngừng phục vụ bảng phụ đó. Trước khi nó thực sự chuyển bảng phụ đi, máy chủ phụ lại làm một quá trình nén rất nhanh khác để loại trừ bất kì trạng thái không nén chặt trong bản ghi của máy chủ phụ mới đến trong khi quá trình nén trước đang xảy ra. Sau khi quá trình nén thứ hai hoàn tất, bảng phụ có thể được tải trên một máy chủ phụ khác mà không cần yêu cầu bất kì sự khôi phục bản ghi nào.

Khai thác tính bất biến

Bên cạnh bộ đệm SStable, nhiều phần khác của hệ thống Bigtable được làm đơn giản hóa mặc dù thực tế là tất cả các phần của Bigtable mà chúng ta tạo ra đều là bất biến. Ví dụ, chúng ta không cần phải đồng bộ hóa việc truy xuất vào hệ thống file khi đọc từ SStable. Như một kết quả tất yếu, kiểm soát trùng hợp qua các hàng có thể được thực

hiện rất hiệu quả. Chỉ những cấu trúc dữ liệu không bền vững mà bị truy xuất bởi cả quá trình đọc và viết mới là memtable. Để giảm sự tranh chấp trong khi đọc từ memtable, chúng ta tạo mỗi hàng memtable mới “bản sao ghi” (copy-on-write) và cho phép việc đọc và ghi được xử lý song song.

SStable là bất biến, vấn đề xóa bỏ những dữ liệu đã bị xóa được biến thành việc tập hợp những dữ liệu Sstable cũ và không phù hợp. Mỗi Sstable của bảng phụ được đăng ký trong bảng Metadata. Máy chủ chính xóa bỏ các SStalbe cũ trên tập các Sstable [14], nơi mà bảng Metadata chứa các thiết lập của root.

2.7. Ước lượng hiệu năng

Theo [11], một cụm Bigtable được cài đặt với N máy chủ phụ để phân phối hiệu năng và khả năng mở rộng của Bigtable. Máy chủ phụ được cấu hình để sử dụng 1GB bộ nhớ và để ghi vào các ổ GFS bao gồm có 1786 máy với 2 ổ cứng IDE 400 GB. Những máy này được sắp xếp theo dạng cây 2 cấp, băng thông chung xấp xỉ 100-200 Gbps tại root. Tất cả các máy đều có chung điều kiện máy chủ và vì thế thời gian đi vòng giữa 1 cặp máy bất kì nào đó đều nhỏ hơn 2 mili giây.

R là số khóa hàng riêng biệt của Bigtable liên quan đến kiểm tra. R được chọn sao cho mỗi chuẩn đọc hoặc ghi đều xấp xỉ 1GB dữ liệu trên mỗi máy chủ phụ.

Chuẩn ghi tuần tự sử dụng khóa hàng với tên từ 0 tới $R-1$. Khoảng cách giữa các khóa hàng được chia vào $10N$ dải bằng nhau. Những dải này được gán cho N client bởi 1 bộ lập lịch trung tâm, bộ này sẽ gán dải sẵn sàng kế tiếp cho 1 client sớm nhất khi client kết thúc tiến trình mà dải trước đó đã gán cho nó. Việc gán động này giúp giảm bớt tác động của các thay đổi về hiệu năng gây ra bởi các tiến trình khác đang chạy trên máy khách. Mỗi khóa hàng được ghi 1 xâu đơn. Mỗi xâu được sinh ngẫu nhiên và do đó không thể nén được. Thêm vào đó, các xâu bên dưới khóa hàng khác nhau là khác nhau, vì thế không thể nén liên hàng (cross-row). Chuẩn ghi ngẫu nhiên cũng tương tự ngoại trừ việc khóa hàng được bấm theo modul R ngay tức thì trước khi được ghi, vì thế hoạt động ghi được trải rộng đều qua toàn bộ khoảng trống giữa các hàng trong suốt quá trình ghi.

Chuẩn đọc liên tiếp sinh ra khóa hàng bằng chính xác cách mà chuẩn ghi liên tiếp, nhưng thay vì ghi dưới khóa hàng, nó đọc xâu lưu trữ bên dưới khóa hàng. Tương tự, chuẩn đọc ngẫu nhiên hoạt động giống như chuẩn ghi ngẫu nhiên.

Chuẩn scan tương tự như chuẩn đọc tuần tự, nhưng sử dụng hỗ trợ cung cấp bởi Bigtable API để scan tất cả các giá trị trong dải hàng. Sử dụng một bộ scan giảm số RPC được thực thi bởi chuẩn này bởi 1 RPC đơn lẻ đem về một chuỗi lớn giá trị từ máy chủ phụ.

Hình dưới cho thấy kết quả đánh giá về hiệu năng của các chuẩn khi đọc và ghi 1000 B dữ liệu. Bảng cho thấy số hoạt động trên một giây của một máy chủ phụ, và đồ thị cho thấy tổng số hoạt động trên 1 máy chủ phụ.

Experiment	# of Tablet Servers			
	1	50	250	500
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843

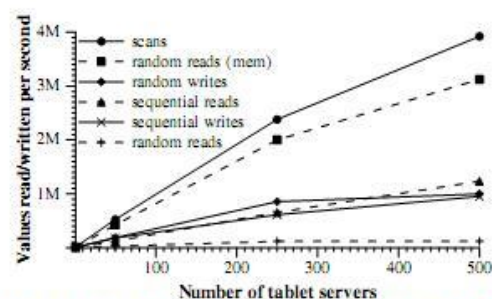


Figure 6: Number of 1000-byte values read/written per second. The table shows the rate per tablet server; the graph shows the aggregate rate.

Hình 3 : Số lần đọc và ghi trên 1 giây với 1000 byte dữ liệu.

Hiệu năng của 1 máy chủ phụ

Chúng ta sẽ cân nhắc đến hiệu năng của 1 máy chủ phụ đơn lẻ. Đọc ngẫu nhiên chậm hơn tất cả các hoạt động khác. Mỗi hoạt động đọc ngẫu nhiên gồm có sự chuyển giao của 64KB block Sstable qua mạng từ GFS tới một máy chủ phụ, trong khi chỉ có 1000 byte là được sử dụng. Máy chủ phụ thực thi xấp xỉ 1200 phép đọc trong 1 giây, dịch xấp xỉ 75MB/s dữ liệu từ GFS. Bảng thông này đủ để làm đầy các chip của máy chủ phụ vì những chi phí cho ngăn xếp của mạng, phân tích cú pháp Sstable, mã Bigtable, và cũng hầu như đủ để làm đầy các kết nối mạng trong hệ thống. Hầu hết các ứng dụng Bigtable với kiểu mà một mẫu truy cập giảm kích thước block xuống nhỏ hơn, điển hình là 8KB.

Đọc ngẫu nhiên từ bộ nhớ nhanh hơn nhiều bởi cứ mỗi phép đọc 1000Byte được đáp ứng từ máy chủ phụ địa phương mà không cần tìm về các block 64KB từ GFS.

Ghi ngẫu nhiên và tuần tự thực thi tốt hơn là đọc ngẫu nhiên vì mỗi máy chủ phụ nối tất cả những phép ghi tiếp theo vào một bản ghi thực thi và sử dụng nhóm thực thi để ghi hiệu quả. Không có khác biệt nhiều về hiệu năng giữa ghi ngẫu nhiên và ghi tuần tự; trong cả hai trường hợp, tất cả các phép ghi vào máy chủ phụ đều được ghi lại vào cùng bản ghi thực thi. Đọc tuần tự tốt hơn đọc ngẫu nhiên do mỗi block Sstable 64KB lấy về từ GFS được lưu vào trong một bộ đệm block, nơi mà nó được sử dụng để phục vụ cho 64 yêu cầu đọc tiếp theo.

Chia tỷ lệ

Thông lượng chung tăng đột ngột, thậm chí theo hệ số hàng trăm, và chúng ta tăng số máy chủ phụ trong 1 hệ thống từ 1 lên 500. Ví dụ, hiệu năng của đọc ngẫu nhiên từ bộ nhớ tăng theo hệ số 300, số máy chủ phụ phải tăng theo hệ số 500. Cách xử lý này xảy ra do tình trạng nghẽn cổ chai trong khi thực thi cho mỗi chuẩn này là CPU máy chủ phụ riêng lẻ.

Tuy nhiên, hiệu năng không tăng theo tuyến. Với hầu hết các chuẩn, có 1 sự giảm đáng kể trong thông lượng qua mỗi máy chủ khi đi từ máy chủ phụ 1 đến 50. Sự giảm này gây ra bởi sự thiếu cân bằng trong tải của cấu hình các máy chủ, thường do các tiến trình khác nhau tranh chấp CPU và mạng. Thuật toán cân bằng tải của chúng ta cố gắng làm việc với sự mất cân bằng, nhưng không thể gọi là hoàn hảo do 2 lý do chính: việc cân bằng lại bị làm nghẹt để giảm số lần di chuyển băng phụ (một băng phụ không sẵn sàng trong một khoảng thời gian ngắn, thường nhỏ hơn 1s, khi nó bị di chuyển), và tải được sinh ra bởi các chuẩn di chuyển xung quanh theo sự phát triển của chuẩn đó.

Chuẩn đọc ngẫu nhiên cho thấy sự chia tỷ lệ tệ nhất (thông lượng chung tăng theo hệ số 100 trong khi số máy chủ phải tăng theo hệ số 500). Nguyên nhân là do chúng ta chuyển 1 block lớn 64KB qua mạng cho mỗi phép đọc 1000byte. Sự truyền này làm đầy kết nối 1Gigabit trong mạng, kết quả là thông lượng trên mỗi máy chủ giảm đáng kể và chúng ta phải tăng số máy chủ.

Chương 3: Hệ thống quản lý file phân tán Hadoop

3.1. Khái niệm cơ bản về hệ thống Hadoop

Hadoop là một framework Java hỗ trợ các ứng dụng phân tán dữ liệu. Nó cho phép các ứng dụng làm việc với hàng ngàn node và hàng petabytes dữ liệu. Hadoop bắt nguồn từ Google's MapReduce và Google File System (GFS).

Hadoop là một dự án Apache nhận được nhiều sự đóng góp của các công ty lớn như Yahoo, Google, IBM. Yahoo sử dụng Hadoop cho giải pháp lưu trữ các trang web tìm kiếm và sử dụng cho các doanh nghiệp quảng cáo của mình. Hadoop đã được đưa vào trong các khóa học về lập trình phân tán trong trường đại học dưới sự hỗ trợ của Google.

Hadoop được khởi xướng bởi Doug Cutting. Với mục tiêu ban đầu được phát triển là nhằm hỗ trợ việc phân phối cho các dự án máy tìm kiếm Nutch[12].

3.1.1. Kiến trúc của Hadoop

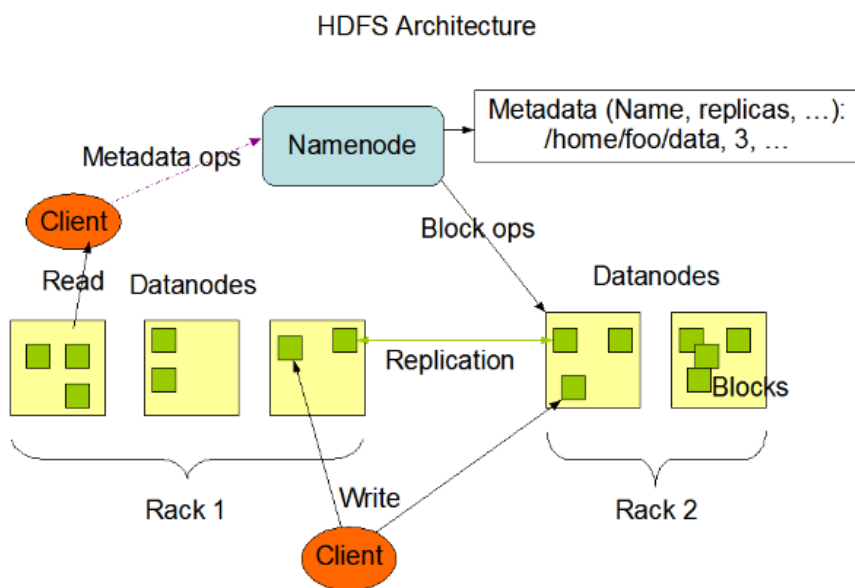
Hadoop bao gồm có nhân Hadoop, nhân này cung cấp truy cập tới file hệ thống mà Hadoop hỗ trợ. “Rack awareness” là một tối ưu hóa sẽ đưa vào tài khoản các cụm địa lý của các server, lưu lượng mạng giữa các server tại các cụm địa lý khác nhau được giảm thiểu. Tính đến tháng 6 năm 2008, danh sách các file hệ thống được hỗ trợ bao gồm:

- HDFS: hệ thống tập tin riêng của Hadoop. Nó được thiết kế để co giãn tới hàng petabytes lưu trữ và chạy trên file hệ thống của hệ điều hành nằm bên dưới.
- File hệ thống Amazon S3
- Cloud Store: giống HDFS
- FTP Filesystems.
- Read-only HTTP và các hệ thống file HTTPS

Hệ thống file HDFS lưu trữ các tập tin lớn (kích thước file lý tưởng là bội số của 64M), qua nhiều máy tính phức tạp. Nó đạt được độ tin cậy bằng cách sao chép dữ liệu trên nhiều máy chủ, và vì thế không yêu cầu lưu trữ RAID trên máy chủ. Với giá trị sao chép mặc định, dữ liệu được lưu trữ trên 3 node: 2 trên cùng một rack, và 1 ở rack khác.

Hệ thống file được xây dựng từ một cụm node dữ liệu, từng node cung cấp lên các khối dữ liệu qua mạng sử dụng một giao thức cụ thể nào đó tới HDFS. Chúng cũng cung cấp dữ liệu qua HTTP, cho phép truy cập tới tất cả nội dung từ một web browser hoặc một client nào đó. Các node dữ liệu có thể giao tiếp với các node khác để tái cân bằng dữ liệu, di chuyển các bản sao vòng quanh, và để giữ bản sao của các dữ liệu quan trọng.

Một filesystem yêu cầu một máy chủ duy nhất, Namenode. Đây là điểm không thích hợp duy nhất trong việc cài đặt HDFS. Nếu NameNode tắt, filesystem sẽ tắt. Khi nó trở lại, NameNode sẽ phát lại tất cả các hoạt động nổi bật. Quá trình phát lại này có thể mất hơn nửa giờ cho một cụm lớn. Filesystem bao gồm cả Secondary NameNode, thứ mà làm cho một vài người nghĩ rằng khi NameNode chính offline, NameNode thứ 2 sẽ thay thế. Trong thực tế, NameNode thứ 2 này kết nối một cách thường xuyên tới các namenode và tải một bản chụp thông tin thư mục của NameNode chính, mà sau đó sẽ được lưu vào thư mục. Namenode thứ 2 được sử dụng cùng với các bản ghi chỉnh sửa của NameNode chính để tạo ra một cấu trúc thư mục cập nhật.



Hình 4: Kiến trúc tổng thể của Hadoop

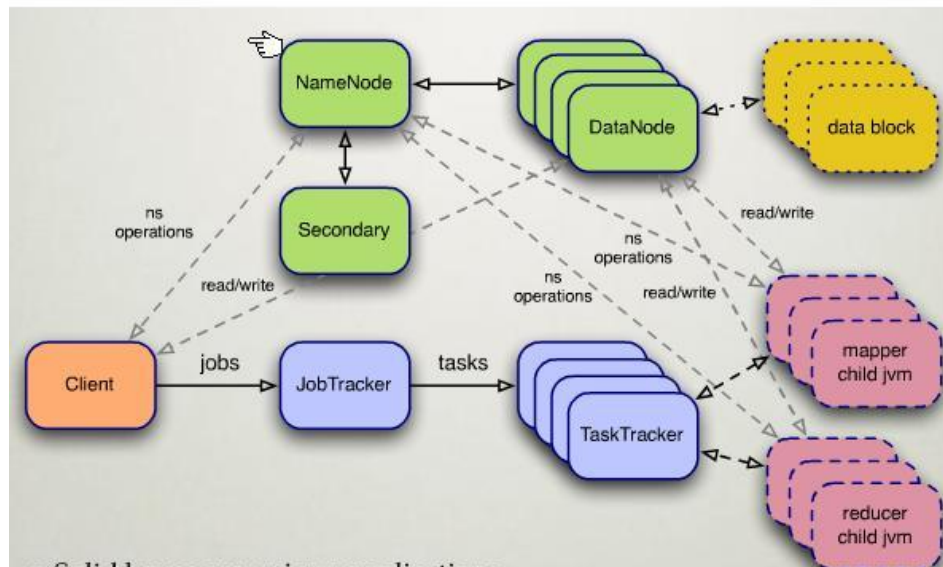
Một hạn chế của HDFS là nó không thể được mount trực tiếp bởi hệ điều hành hiện tại. Lấy dữ liệu vào và ra khỏi hệ thống file HDFS, một hoạt động cần phải được làm

trước và sau khi thực hiện một công việc, có thể gây ra bất tiện. Một filesystem trong UserSpace đã được phát triển để giải quyết vấn đề này, ít nhất là cho Linux và một số hệ thống Unix khác.

Sao chép dữ liệu 3 lần là rất tốn kém. Để giảm bớt chi phí, phiên bản gần đây của HDFS đã xóa việc hỗ trợ mã hóa, theo đó nhiều khối của cùng một tập tin được kết hợp để tạo ra một khối chẵn lẻ. HDFS tạo ra các khối chẵn lẻ không đồng bộ và sau đó giảm các nhân tố của việc sao chép từ 3 xuống còn 2. Các nghiên cứu đã chỉ ra rằng kỹ thuật này giảm các yêu cầu lưu trữ vật lý từ một nhân tố từ 3 xuống còn khoảng 2.2.

3.1.2. Job Tracker và Task Tracker: các máy MapReduce

Trên các file hệ thống có máy MapReduce, trong đó bao gồm một Job Tracker, mà các ứng dụng client xác nhận các công việc MapReduce. Job Tracker đẩy các công việc ra các node Task Tracker trong cụm, cố gắng để giữ cho công việc gần với dữ liệu có thể. Với các filesystem rack-aware, Job Tracker biết được node chứa dữ liệu và máy nào gần nó. Nếu công việc không thể được lưu trữ trên các node dữ liệu thực tế, ưu tiên sẽ được dành cho các node trong cùng một rack. Điều này làm giảm lưu lượng mạng trên trục chính. Nếu một Task Tracker lỗi hoặc bị time out, phần đó của công việc sẽ được lập lịch lại. Nếu Job Tracker lỗi, tất cả các công việc đang thực thi sẽ bị mất.



Hình 5: Các máy MapReduce

Hadoop 0.21 có thêm một số trạm kiểm soát tới quá trình này, các Job Tracker ghi lại những gì được đưa lên filesystem. Khi một Job Tracker khởi động, nó sẽ chờ bất kì dữ liệu nào, để nó có thể khởi động lại làm việc từ khi nó rời khỏi. Trong các phiên bản trước đó của Hadoop, tất cả các công việc đang hoạt động sẽ bị mất khi Job Tracker khởi động lại.

Những hạn chế của phương pháp này là:

- Việc phân chia công việc cho các Task Tracker là đơn giản. Mỗi Task Tracker có một số khe có sẵn (ví dụ như 4 khe). Mỗi map hoạt động hoặc giảm task chiếm một khe. Các Job Tracker phân bổ công việc tới các tracker gần với dữ liệu nhất với một khe có sẵn. Không có cân nhắc tới các hoạt động tải hiện thời của các máy đã được chỉ định.
- Nếu một Task Tracker quá chậm, nó có thể làm trì hoãn toàn bộ các hoạt động MapReduce, đặc biệt là về khoảng cuối của công việc, khi mà tất cả mọi thứ chỉ còn chờ một task chậm duy nhất.

Filesystem HDFS không bị giới hạn bởi các công việc MapReduce. Nó có thể được sử dụng cho các ứng dụng khác, nhiều trong số chúng là trên nền Apache. Danh sách bao

gồm cơ sở dữ liệu Hbase, hệ thống học máy Apache Mahout, và hoạt động của ma trận. Về mặt lý thuyết, Hadoop có thể được sử dụng cho bất kỳ công việc nào mà tính định hướng khối mạnh hơn là thời gian thực, dữ liệu tập trung, và có khả năng làm việc trên các mảnh dữ liệu song song.

3.2. Cơ chế MapReduce

3.2.1. Giới thiệu

MapReduce là một framework được giới thiệu bởi Google[8] để hỗ trợ các máy tính phân tán trên các bảng dữ liệu lớn đặt trên các cụm máy tính.

Framework được lấy ý tưởng từ Map và giảm các hàm thường được sử dụng trong lập trình chức năng. Thư viện MapReduce được viết bằng C++, C#, Erlang, Java, Python, F#, R và các ngôn ngữ khác.

MapReduce được dùng cho xử lý các bảng dữ liệu khổng lồ trên một số nhóm của các vấn đề có thể phân phối sử dụng một số lượng lớn máy tính (node), gọi chung lại là một cụm (cluster). Xử lý điện toán có thể xảy ra trên dữ liệu được lưu trữ hoặc trên filesystem (không có cấu trúc) hoặc bên trong một cơ sở dữ liệu (có cấu trúc).

- Quá trình “Map”: các node điều khiển nắm đầu vào input, chia nó ra thành các vấn đề con nhỏ hơn, và phân phối nó cho các node làm việc. Một node thực thi có thể làm điều này một lần nữa, dẫn đến một cấu trúc cây đa cấp.

Các node thực thi xử lý các vấn đề nhỏ hơn, và đẩy kết quả về cho node điều khiển của nó.

- Quá trình “Reduce”: Node điều khiển lấy kết quả của tất cả các vấn đề con và tập hợp chúng lại trên đường ra output- kết quả cho vấn đề ban đầu.

Ưu điểm của MapReduce là nó cho phép xử lý phân tán các ánh xạ và các hoạt động rút gọn. Cung cấp cho mỗi hoạt động ánh xạ là độc lập với những hoạt động khác, tất cả các map có thể được thực hiện song song – mặc dù trong thực tế thì nó bị giới hạn bởi bởi nguồn dữ liệu và số CPU gần dữ liệu. Tương tự, một tập hợp các “reducer” có thể thực hiện giai đoạn giảm thiểu – tất cả những gì được yêu cầu là tất cả output của hoạt động ánh xạ mà chia sẻ cùng một khóa được trình bày với cùng một reducer, tại cùng một thời

điểm. Trạng thái song song cũng cung cấp khả năng phục hồi từ những lỗi cục bộ của server hoặc lưu trữ trong quá trình hoạt động: nếu một mapper hoặc một reducer bị lỗi, công việc có thể được tái lập lịch – giả định rằng dữ liệu đầu vào là sẵn có.

3.2.2. Các thành phần logic

3.2.2.1. Map

Cả chức năng Map và chức năng Reduce của MapReduce đều được định nghĩa đối với các dữ liệu được cấu trúc theo cặp (khóa, giá trị). Map lấy một cặp dữ liệu với một loại trong một tên miền dữ liệu, và trả lại một danh sách các cặp trong một tên miền khác:

$\text{Map}(k1, v1) \rightarrow \text{list}(k2, v2)$

Chức năng Map được áp dụng song song với mọi mục trong bảng dữ liệu input. Điều này tạo ra một danh sách các cặp $(k2, v2)$ cho mỗi lần gọi. Sau đó, MapReduce framework thu thập tất cả các cặp với cùng một khóa từ tất cả danh sách và nhóm chúng lại với nhau, tạo một nhóm cho mỗi một trong các key đã có.

Số ánh xạ thường phụ thuộc vào kích thước tổng thể của input, hay nói cách khác, là tổng số block của các file input. Mật độ phù hợp nằm trong khoảng 10-100 ánh xạ trên node.

3.2.2.2. Reduce

Chức năng Reduce sau đó được áp dụng song song với mỗi nhóm, do đó tạo ra một bộ sưu tập các giá trị trong cùng một tên miền:

$\text{Reduce}(k2, \text{list}(v2)) \rightarrow \text{list}(v3)$

Mỗi lần gọi Reduce thường cho ra kết quả là một giá trị $v3$ hoặc là giá trị rỗng, mặc dù một lần gọi được cho phép trả về nhiều hơn một giá trị. Kết quả trả về của tất cả các lần gọi được tập hợp lại thành danh sách kết quả mong muốn.

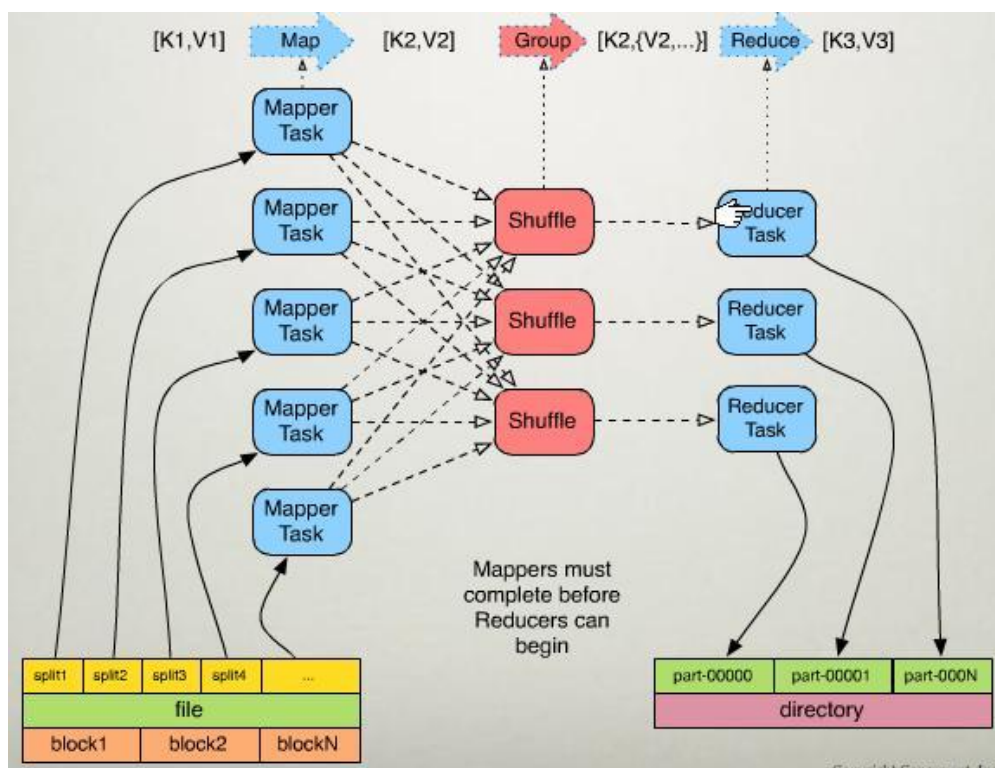
Một Reducer có 3 giai đoạn chính: Shuffle, Sort và Reduce

Shuffle: Input tới Reducer là output của các Mapper đã được sắp xếp lại. Ở giai đoạn này, framework sẽ lấy về những phân vùng có liên quan với output của tất cả các mapper, thông qua HTTP.

Sort: Framework nhóm các đầu vào Reducer bằng các khóa. Giai đoạn này hoạt động đồng thời với giai đoạn Shuffle.

Reduce: trong giai đoạn này, các phương thức sẽ được gọi cho mỗi cặp (khóa, danh sách giá trị) trong input đã được chia nhóm. Output của tác vụ Reduce được ghi lên hệ thống file thông qua hàm `OutputCollector.collect (WritableComparable, Writable)`. Ứng dụng có thể sử dụng bộ báo cáo Reporter để báo cáo tiến trình, cài đặt tin nhắn trạng thái cấp độ của ứng dụng và cập nhật bộ Counter. Dữ liệu đầu ra của Reducer không được sắp xếp.

Theo cách đó, MapReduce biến đổi một danh sách các cặp (khóa, giá trị) thành một danh sách các giá trị. Hành vi này là khác nhau từ map lập trình chức năng và giảm thiểu sự kết hợp, mà chấp nhận một danh sách các giá trị tùy ý và trả về một giá trị duy nhất kết hợp tất cả các giá trị được trả về bởi Map.



Hình 6: Thành phần logic Mapper và Reducer

Điều này là cần thiết nhưng không đủ khả năng để có sự thực thi của map và giảm thiểu sự trừu tượng nhằm thực hiện MapReduce. Hơn nữa việc triển khai có hiệu quả MapReduce yêu cầu một hệ thống file phân tán để kết nối tới các giai đoạn tiến trình thực hiện Map và Reduce.

Một ví dụ điển hình trong ứng dụng của MapReduce là xử lý đếm số lần xuất hiện của các từ khác nhau trong một văn bản cho trước:

```
void map(String name, String document):  
    // name: document name  
    // document: document contents  
    for each word w in document:  
        EmitIntermediate(w, 1);  
  
reduce(String word, Iterator partialCounts):  
    // word: a word  
    // partialCounts: a list of aggregated partial counts  
    int result = 0;  
    for each pc in partialCounts:  
        result += ParseInt(pc);  
    Emit(result);
```

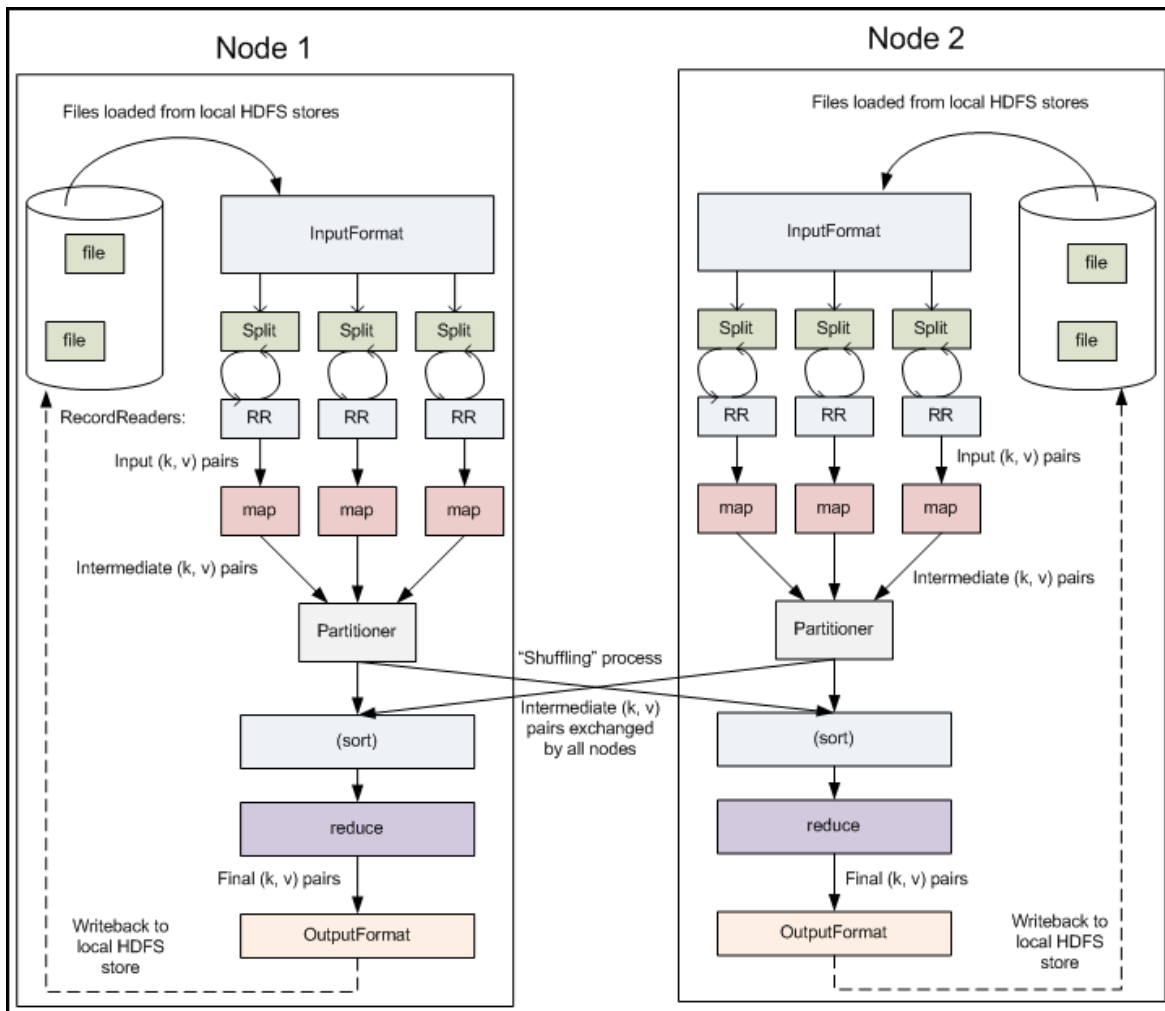
Ở đây, mỗi văn bản được chia thành các từ, và mỗi từ được đếm lần đầu tiên với giá trị “1” bởi chức năng Map, sử dụng từ đó như là khóa. Framework sẽ đẩy tất cả các cặp với cùng một khóa và gọi chúng cùng một tên tới Reduce, theo đó, Reduce sẽ cần tính tổng của tất cả các giá trị input để tìm ra tổng số lần xuất hiện của từ.

3.2.3. Mô hình luồng dữ liệu

Dữ liệu đi lần lượt qua các bộ sau:

- Một trình đọc đầu vào (tương ứng trong hình là khối InputFormat)
- Một chức năng Map (khối map)

- Một chức năng phân vùng (khối Partitioner)
- Một chức năng reduce (khối reduce)
- Một trình ghi đầu ra (OutputFormat)



Hình 7: Sơ đồ luồng dữ liệu

Trình đọc đầu vào: chia dữ liệu vào thành các mảnh từ 16M đến 128MB và framework sẽ gán mỗi phần cho mỗi chức năng Map. Bộ đọc đầu vào đọc dữ liệu từ các kho ổ định và sinh ra các cặp khóa/ giá trị.

Một ví dụ phổ biến là đọc một thư mục đầy đủ của tập tin văn bản và trả lại từng dòng như là một bản ghi.

Chức năng Map: mỗi chức năng Map lấy một loạt các cặp khóa/dữ liệu, xử lý từng cặp, sinh ra 0 hoặc nhiều hơn các cặp khóa/dữ liệu output. Các kiểu đầu vào và đầu ra của map có thể khác nhau từng cặp một .

Nếu ứng dụng đang đếm từ, chức năng Map sẽ phá vỡ đường tới các từ và ở đầu ra thì từ được coi như khóa và '1' là giá trị.

Chức năng phân vùng: Đầu ra của tất cả các Mapper được giao cho một Reducer cụ thể bởi chức năng phân vùng của ứng dụng. Chức năng phân vùng được nhận khóa và số reducer và trả lại chỉ số mong muốn được giảm thiểu.

Chức năng reduce: Framework gọi chức năng reduce của ứng dụng một lần cho mỗi khóa duy nhất theo thứ tự đã sắp xếp. Việc giảm thiểu có thể được lặp lại qua các giá trị được liên kết với khóa đó 0 hoặc nhiều hơn giá trị đầu ra.

Trình ghi đầu ra: ghi kết quả output tới các kho lưu trữ ổn định, thường là các hệ thống file phân tán, ví dụ như Google File System.

3.2.4. Đánh giá

MapReduce đạt được tính tin cậy bằng cách chia các hoạt động cho mỗi node trong mạng. Mỗi node được mong đợi sẽ báo cáo lại một cách định kì với công việc đã hoàn thành và trạng thái cập nhật. Nếu một node im lặng trong một khoảng thời gian, node điều khiển (tương tự như server điều khiển trong Google File System) sẽ ghi lại nút này như là đã chết và gửi công việc được giao cho nó đến một node khác. Các hoạt động cá nhân sử dụng các hoạt động nguyên tử để đặt tên file đầu ra như là một kiểm tra để bảo đảm không có những chuỗi xung đột song song đang chạy. Khi 1 file được đặt lại tên, có thể copy chúng thành một tên khác ngoài tên của nhiệm vụ.

Hoạt động giảm thiểu vận hành theo cùng một cách. Bởi vì các đặc tính kém hơn liên quan đến các hoạt động song song, node điều khiển cố gắng lập lịch các hoạt động giảm thiểu trên cùng một node, hoặc trên cùng một rack. Đặc tính này là rất quý bởi vì nó giữ gìn băng thông đi qua trục mạng chính của trung tâm dữ liệu.

MapReduce có ích trong một loạt các ứng dụng và kiến trúc, bao gồm “phân phối grep, phân loại, biểu đồ liên kết web đảo ngược, giới hạn vector trên máy chủ, phân cụm tài liệu, học máy, thống kê máy dịch thuật.....”. Đáng kể nhất, khi MapReduce hoàn tất, nó đã được sử dụng để phục hồi hoàn toàn chỉ mục của Google của World Wide Web, và thay thế các chương trình đặc biệt dùng để cập nhật chỉ số và chạy các phân tích khác nhau.

Đầu vào và đầu ra ổn định của MapReduce thường được lưu trữ trong một hệ thống file phân tán. Dữ liệu tạm thời thường được lưu trữ trên đĩa và được lấy từ xa.

3.3. Ứng dụng của Hadoop

3.3.1. Hadoop trong máy tìm kiếm Yahoo

Tháng 2 năm 2008, Yahoo đã khởi động sản phẩm ứng dụng Hadoop mà họ khẳng định là lớn nhất trên thế giới. Bản đồ Web tìm kiếm của Yahoo là một ứng dụng Hadoop chạy trên hơn 10000 cụm nhân Linux và tạo ra những dữ liệu mà ngày nay được sử dụng trong mọi truy vấn tìm kiếm web của Yahoo.

Có nhiều cụm Hadoop đồng thời trên Yahoo, mỗi cụm chiếm giữ một trung tâm dữ liệu duy nhất. Không có một hệ thống file HDFS hay cơ chế MapReduce nào chia cắt các trung tâm dữ liệu này, thay vì mỗi trung tâm có một hệ thống file riêng biệt. Những cụm server chạy trên Linux, và được cấu hình lúc khởi động sử dụng KickStart. Tất cả các máy khởi động ảnh Linux, bao gồm cả việc phân bố Hadoop. Cấu hình các cluster được trợ giúp thông qua một chương trình gọi là ZooKeeper. Công việc mà các cluster thực hiện được biết đến là để bao hàm việc tính toán các chỉ số cho các máy tìm kiếm Yahoo.

3.3.2. Hadoop trên các dịch vụ Amazon EC2/S3

Có thể chạy Hadoop trên Amazon Elastic Compute Cloud (EC2) và Amazon Simple Storage Service (S3). Có hỗ trợ cho filesystem S3 trong sự phân phối của Hadoop. Và Hadoop tạo ra hình ảnh máy EC2 sau khi phát hành. Từ quan điểm hiệu suất tối đa, Hadoop trên EC2/S3 là không hiệu quả, filesystem S3 điều khiển từ xa và bị trễ từ mọi hoạt động ghi cho đến ghi dữ liệu được đảm bảo không bị mất. Điều này loại bỏ các lợi thế lên lịch công việc gần với dữ liệu để tiết kiệm tải mạng của Hadoop.

3.3.3. Hadoop với Sun Grid Engine

Hadoop cũng có thể được sử dụng trong các môi trường tính toán hiệu năng cao. Tích hợp với Sun Grid Engine đã phát hành, và chạy Hadoop trên Sun Grid là có thể. Lúc ban đầu, bộ lập lịch thời gian CPU không có khái niệm về vị trí của dữ liệu. một tính năng chính của Hadoop Runtime “làm việc trên cùng một server hoặc rack như là dữ liệu” vì thế đã không còn.

Sun cũng có những dự án Hadoop Live CD OpenSolaris, cho phép chạy một cụm Hadoop đầy đủ chức năng sử dụng đĩa Cd.

Chương 4: Kiến trúc HBase

4.1. Giới thiệu HBase

Hbase là một hệ mã nguồn mở cung cấp một hệ thống lưu trữ giống như Bigtable cho các môi trường tính toán phân tán Hadoop.

Dữ liệu được sắp xếp một cách logic vào các bảng, hàng và cột. Một giao diện kiểu biến lập sẵn sàng cho việc scan qua các dãy hàng, dĩ nhiên, có khả năng lấy được một giá trị cột cho một khóa hàng. Bất kì cột riêng lẻ nào đều có thể có nhiều phiên bản cho cùng một khóa hàng.

4.2. Mô hình dữ liệu

Hbase sử dụng một mô hình dữ liệu rất giống với Bigtable. Các ứng dụng lưu trữ các hàng dữ liệu trong các bảng được gán nhãn. Một hàng dữ liệu có khóa hàng có thể phân loại và một số lượng tùy ý các cột. Bảng được lưu trữ rải rác, vì vậy các hàng trong cùng một bảng có thể có số lượng cột rất khác nhau.

Tên cột có định dạng: “<tên họ cột>:<nhãn>”, tên họ cột và nhãn có thể là các mảng dữ liệu bất kì. Một bảng làm cho có hiệu lực các thiết lập của các họ cột của nó. Việc điều chỉnh thiết lập của các họ cột được hoàn thành bởi các hoạt động quản trị hiệu năng trên bảng. Tuy nhiên, những nhãn mới có thể được sử dụng ở bất kì một hoạt động ghi nào mà không cần thông báo cho nó.

Chỉ một hàng trong một thời điểm có thể bị khóa. Phép ghi các hàng luôn luôn là nguyên tử, nhưng cũng có khả năng khóa một hàng và thực hiện đồng thời cả ghi và đọc trên hàng đó một cách tự động.

Phần mở rộng được thêm vào để cho phép khóa nhiều hàng, nhưng nó không phải là thiết lập chuẩn và phải được cho phép một cách rõ ràng.

4.2.1. Khung nhìn khái niệm

Một cách khái niệm, một bảng có thể được hiểu là một tập hợp các hàng được định vị bởi một khóa hàng (và nhãn thời gian) và nơi mà cột bất kì có thể có một giá trị cho

các khóa hàng riêng biệt. Ví dụ sau đây là một form đã được thay đổi một chút từ ví dụ ở phần 2.2 (có thêm cột mime)

Row Key	Time Stamp	Column "contents:"	Column "anchor:"		Column "mime:"
"com.cnn.www"	t9		"anchor:cnnsi.com"	"CNN"	
	t8		"anchor:my.look.ca"	"CNN.com"	
	t6	"<html>..."			"text/html"
	t5	"<html>..."			
	t3	"<html>..."			

4.2.2. Khung nhìn lưu trữ vật lý

Mặc dù ở cấp khái niệm, các bảng có thể được nhìn như một tập rải rác các hàng, về phương diện vật lý, chúng được lưu trữ trên cơ sở họ cột. Đây là một lưu ý quan trọng mà các nhà thiết kế lược đồ và ứng dụng phải ghi nhớ.

Một cách hình ảnh, bảng ở trên được lưu trữ như dưới đây

Row Key	Time Stamp	Column "contents:"
"com.cnn.www"	t6	"<html>..."
	t5	"<html>..."
	t3	"<html>..."

Row Key	Time Stamp	Column "anchor:"	
"com.cnn.www"	t9	"anchor:cnnsi.com"	"CNN"

	t8	"anchor:my.look.ca"	"CNN.com"
--	----	---------------------	-----------

Row Key	Time Stamp	Column "mime:"
"com.cnn.www"	t6	"text/html"

Chú ý rằng trong sơ đồ trên những ô trống được chỉ ra trong khung nhìn khái niệm không được lưu trữ bởi vì chúng không cần thiết ở trong một định dạng lưu trữ hướng theo cột (column-oriented). Theo đó, một yêu cầu cho giá trị của cột “contents:” tại thời gian t8 sẽ không có giá trị trả lại. Tương tự, một yêu cầu cho giá trị “anchor.my.look.ca” tại thời gian t9 sẽ không có giá trị trả lại.

Tuy nhiên, nếu nhãn thời gian không được cấp, giá trị gần nhất cho một cột sẽ được trả lại và nó sẽ là giá trị đầu tiên có nhãn thời gian, nhãn thời gian sẽ giảm dần. Theo đó, một yêu cầu giá trị của tất cả các cột trong hàng “com.cnn.www” nếu không nhãn thời gian được chỉ định sẽ là: giá trị của “contents:” từ thời gian t6, giá trị của “anchor.cnnsi.com” từ thời gian t9, giá trị của “anchor.my.look.ca” từ thời gian t8 và giá trị của “mime:” từ t6.

Dải hàng

Đối với một ứng dụng, một bảng xuất hiện như một danh sách bộ dữ liệu được sắp xếp theo khóa hàng tăng dần, tên cột tăng dần và nhãn thời gian giảm dần. Theo phương diện vật lý, các bảng được chia vào các dải hàng được gọi là vùng (region) (tương đương trong Bigtable là các bảng phụ). Mỗi dải hàng chứa các hàng từ khóa bắt đầu tới khóa kết thúc, Hbase nhận biết một dải hàng bằng tên bảng và khóa bắt đầu.

Mỗi họ cột trong một vùng được quản lý bởi một HStore. Mỗi HStore có thể có một hoặc nhiều MapFile (một kiểu file Hadoop HDFS) tương tự như Google SStable. Giống như SStable, Mapfile không thể thay đổi khi đã đóng lại. Mapfile được lưu trữ trong HDFS. Những chi tiết khác tương tự, chỉ trừ:

- Mapfile không thể ánh xạ vào bộ nhớ.
- Mapfile duy trì những chỉ số rải rác trong một file riêng lẻ đúng hơn là cuối một file như là SSTable làm.

HBase kế thừa Mapfile vì thế một bộ lọc Bloom có thể được sử dụng để tăng cường hiệu năng tra cứu. Hàm băm đã sử dụng được phát triển bởi Bob Jenkins.

4.3. Kiến trúc và thực thi

Có 3 thành phần chính trong kiến trúc Hbase:

1. HBaseMaster (tương đương máy chủ chính Bigtable)
2. HRegionServer (tương đương máy chủ phụ Bigtable)
3. HBase client, định nghĩa bởi `org.apache.hadoop.hbase.client.Htable`

4.3.1. HBaseMaster

HBaseMaster có trách nhiệm chỉ định các vùng cho các HregionServer . Vùng đầu tiên được chỉ định là vùng Root, là vùng định vị tất cả các vùng Meta được chỉ định. Mỗi vùng Meta ánh xạ 1 số vùng người dùng bao gồm nhiều bảng mà một HBase phục vụ. Một khi tất cả các vùng Meta đã được chỉ định, máy chủ chính sẽ chỉ định các vùng người dùng tới các HregionServer, cố gắng cân bằng số vùng được phục vụ bởi mỗi HregionServer.

Nó cũng giữ một con trỏ tới HregionServer làm host cho vùng Root.

HbaseMaster cũng giám sát tình trạng của mỗi HRegionServer, và nếu nó phát hiện một HRegionServer không thể kết nối tới được, nó sẽ chia cắt bản ghi write-ahead của HRegionServer đó, do đó không có bản ghi write-ahead cho các vùng mà HRegionServer đó phục vụ. Sau khi nó hoàn thành xong việc này, nó sẽ chỉ định lại các vùng đang được phục vụ bởi Hregionserver bị lỗi.

Thêm vào đó, Hbasemaster cũng có trách nhiệm xử lý các hàm quản trị bảng ví dụ như điều khiển trực tuyến/ngoại tuyến của bảng, thay đổi tới các lược đồ bảng(thêm hoặc xóa họ cột)

Không giống Bigtable, hiện nay, khi một Hbasemaster ngừng hoạt động, cụm sẽ tắt. Trong Bigtable, một máy chủ phụ có thể phục vụ các bảng phụ khi kết nối tới máy chủ

chính đã mất. Chúng ta liên kết chúng lại với nhau, bởi chúng ta không sử dụng một hệ thống quản lý khóa bên ngoài giống như Bigtable. Máy chủ chính Bigtable định vị các bảng phụ và một khóa quản lý (Chubby) đảm bảo các truy xuất nguyên tử bởi máy chủ phụ tới các bảng phụ. Hbase sử dụng một con trỏ trung tâm cho tất cả các Hregionserver truy cập: Hbasemaster.

Bảng Meta

Bảng Meta lưu trữ thông tin về mọi vùng người dùng trong Hbase bao gồm một đối tượng Hregioninfo chứa thông tin như các khóa hàng bắt đầu và kết thúc, vùng là trực tuyến hay ngoại tuyến,và địa chỉ của Hregionserver đang phục vụ cho vùng. Bảng Meta có thể mở rộng theo sự mở rộng của các vùng người dùng.

Bảng Root

Bảng Root bị giới hạn vào một vùng đơn lẻ và ánh xạ tất cả các vùng vào bảng Meta. Giống như bảng Meta, nó chứa một đối tượng Hregioninfo cho mỗi vùng Meta và vị trí của Hregionserver đang phục vụ cho vùng Meta đó.

Mỗi hàng trong các bảng Meta và Root có kích thước xấp xỉ 1KB. Kích thước vùng mặc định là 256MB, điều này có nghĩa là vùng Root có thể ánh xạ $2.6 * 10^5$ vùng, ánh xạ tổng cộng $6.9 * 10^{10}$ vùng người dùng, tương đương xấp xỉ $1.8 * 10^{19} \cdot (2^{64})$ byte dữ liệu người dùng.

4.3.2. HRegionServer

Hregionserver có trách nhiệm xử lý các yêu cầu đọc và ghi của phía client. Nó giao tiếp với Hbasemaster để lấy danh sách các vùng để phục vụ và để cho master biết là nó vẫn đang hoạt động.

Các yêu cầu ghi

Khi một yêu cầu ghi tới, đầu tiên nó sẽ được ghi vào bản ghi write-ahead được gọi là Hlog. Tất cả các yêu cầu ghi cho tất cả các vùng mà RegionServer đang phục vụ được ghi vào cùng một bản ghi. Mỗi lần yêu cầu được ghi vào Hlog, nó được lưu trữ trong một bộ đệm gọi là Memcache. Chỉ có một Memcache cho mỗi Hstore.

Các yêu cầu đọc

Các phép đọc được xử lý theo cách kiểm tra Memcache trước tiên và nếu dữ liệu yêu cầu không được tìm thấy, Mapfiles được tìm kiếm cho kết quả.

Cache Flushes

Khi Memcache đạt tới kích thước có thể cấu hình được, nó được đưa vào đĩa, tạo ra một Mapfile mới và một vạch dấu được ghi vào Hlog, do đó khi nó được xem lại, các mục bản ghi trước lần đầy vào cuối cùng được bỏ qua.

Đây bộ đệm xảy ra đồng thời với quá trình xử lý yêu cầu đọc và ghi của regionserver. Trước khi Mapfile mới được di chuyển, các phép đọc và ghi bị treo cho đến khi Mapfile được thêm vào danh sách các Mapfile hoạt động của Hstore.

Nén dữ liệu

Khi số Mapfile vượt quá một ngưỡng có thể cấu hình được, một bộ nén nhỏ được thực thi nhằm củng cố cho Mapfile được ghi gần nhất. Một phép nén lớn được thực thi một cách định kỳ nhằm hợp nhất tất cả các Mapfile vào một Mapfile. Lý do không thực hiện nén lớn thường xuyên là vì mapfile cũ nhất có thể khá lớn và việc đọc và gộp nó với Mapfile cuối cùng, nhỏ hơn rất nhiều, có thể tốn rất nhiều thời gian phụ thuộc vào lượng I/O liên quan đến đọc, gộp và ghi nội dung của Mapfile lớn nhất.

Nén xảy ra đồng thời với quá trình xử lý yêu cầu đọc và ghi của regionserver. Trước khi Mapfile mới được di chuyển, các phép đọc và ghi bị treo cho đến khi Mapfile được thêm vào danh sách các Mapfile hoạt động của Hstore và Mapfile được gộp để tạo ra Mapfile mới đã bị xóa bỏ.

Phân cách vùng

Khi tổng kích thước của Mapfile cho một Hstore đạt tới mức có thể cấu hình được (256MB), một yêu cầu chia tách được gọi. Chia tách vùng chia dải hàng của vùng cha thành một nửa rất nhanh bởi vì các vùng con đọc từ Mapfile của vùng cha.

Vùng cha trở thành ngoại tuyến, RegionServer ghi các vùng con mới vào vùng Meta và master biết rằng một phép chia đã được thực hiện vì thế nó có thể chỉ định các vùng con vào các RegionServer. Nếu tin nhắn chia tách bị mất, master sẽ nhận ra một phép chia đã xảy ra bởi vì nó scan định kỳ vùng Meta cho các vùng đã được chỉ định.

Một khi vùng cha được đóng lại, các yêu cầu đọc và ghi sẽ bị treo. Client có một cơ chế để phát hiện một phép chia vùng và sẽ đợi và thử lại các yêu cầu khi các vùng con trực tuyến.

Khi một phép nén khởi động tại vùng con, dữ liệu từ vùng cha được copy vào vùng con. Khi các vùng con thực thi một phép nén, vùng cha được coi là thừa.

4.3.3. HBase Client

Hbase Client có trách nhiệm tìm kiếm HregionServer đang phục vụ. Cụ thể, Hbase client giao tiếp với Hbasemaster để tìm ra vị trí của vùng Root. Đây chỉ là sự giao tiếp giữa client và master.

Mỗi khi vùng Root được định vị, client liên hệ với regionserver đó và scan vùng Root để tìm ra vùng Meta chứa vị trí của vùng người dùng mà chứa dải hàng mong muốn. Sau đó nó liên hệ với regionserver phục vụ vùng Meta và scan vùng Meta đó để xác định vị trí của vùng người dùng.

Sau khi định vị vùng người dùng, client liên hệ với region server phục vụ vùng đó và giải quyết các yêu cầu đọc và ghi.

Thông tin này được giấu tại client vì thế những yêu cầu tiếp theo không cần phải thông qua tiến trình này.

Chương 5: Cài đặt thực nghiệm và đánh giá hiệu năng

5.1. Môi trường thử nghiệm

Nutch[21] là một hệ tìm kiếm mã nguồn mở, được phát triển trên nền Java và có tổ chức hệ thống file lưu trữ riêng gọi là NFS (Nutch File System). Và Nutch có hỗ trợ cơ chế tích hợp với hệ thống Hadoop. Trong thử nghiệm này tiến hành cài đặt, đánh giá hệ thống file phân tán Hadoop trên máy tìm kiếm Nutch.

Phần cứng: 3 máy tính để bàn desktop

Công cụ, phần mềm:

- Hadoop 0.18.3
- Nutch 0.9
- OpenSSH
- Cygwin

Nội dung thử nghiệm:

- Cài đặt cụm phân tán Hadoop
- Cấu hình Nutch lưu trữ phân tán với cụm Hadoop trên.

5.2. Cài đặt cụm Hadoop phân tán quy mô 3 máy

Sử dụng Cygwin để giả lập môi trường Linux trên các máy để bàn hệ điều hành Window.

a. Cài đặt ssh cho tất cả các máy trong cụm:

- Khởi động dịch vụ ssh: `/usr/sbin/sshd`
- Tạo cặp khóa bí mật, công khai: `ssh-user-config`
- Trao đổi các khóa công khai cho tất cả máy trong cụm:
`cat ~/.ssh/machine1.pub >> ~/.ssh/authorized_keys`

b. Cấu hình Hadoop và Nutch 0.9

- Sửa các file cấu hình trong /hadoop-0.18.3/conf/, bao gồm 3 file hadoop-env.sh, hadoop-site.xml, mapred-default.xml (file này phải tạo mới) và 2 file master & slaves để chỉ định các máy master và slave.

- File hadoop-env.sh. Thêm 2 dòng sau:

```
export HADOOP_HOME=/cygdrive/c/cygwin/nutch-0.9/
```

```
export JAVA_HOME=/cygdrive/c/"Program Files"/Java/jdk1.6.0_12
```

Cách viết /cygdrive/c/... là để nhận được đường dẫn trong Windows.

- File hadoop-site.xml. Thêm các giá trị sau:

```
<property>
  <name>fs.default.name</name>
  <value>hdfs://master:9000</value>
  <description>
    The name of the default file system. Either the literal string
    "local" or a host:port for NDFS.
  </description>
</property>

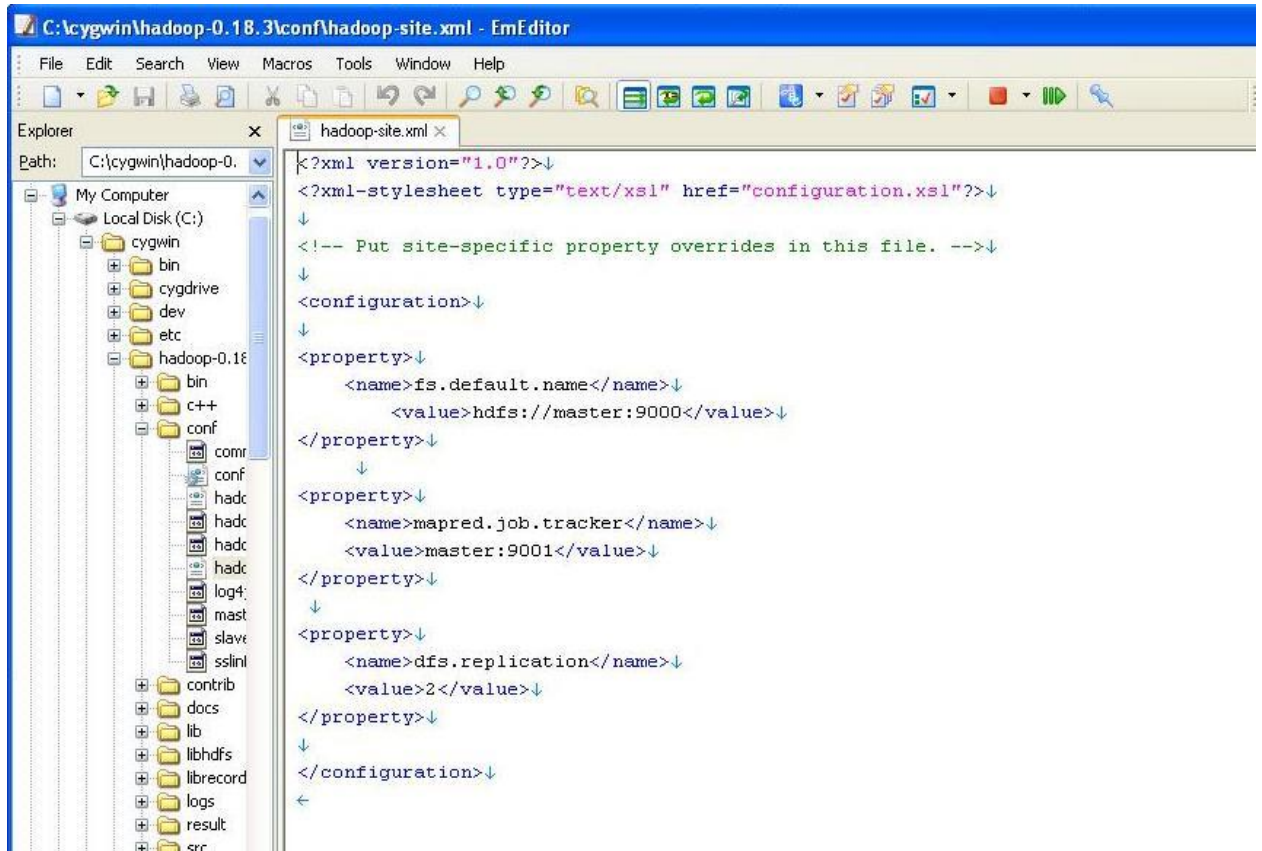
<property>
  <name>mapred.job.tracker</name>
  <value>master:9001</value>
  <description>
    The host and port that the MapReduce job tracker runs at. If
    "local", then jobs are run in-process as a single map and
    reduce task.
  </description>
</property>

<property>
  <name>mapred.tasktracker.tasks.maximum</name>
  <value>2</value>
  <description>
    The maximum number of tasks that will be run simultaneously by
    a task tracker. This should be adjusted according to the heap size
    per task, the amount of RAM available, and CPU consumption of each
    task.
  </description>
</property>

<property>
  <name>mapred.child.java.opts</name>
  <value>-Xmx200m</value>
```

```
<description>
  You can specify other Java options for each map or reduce task here,
  but most likely you will want to adjust the heap size.
</description>
</property>

<property>
<name>dfs.replication</name>
  <value>1</value>
</property>
```



Hình 8: Cấu hình file hadoop-site.xml

- File mapred-default.xml. Thường ko có sẵn, phải tạo mới. Thêm 2 thuộc tính:

```
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
  <property>
    <name>mapred.map.tasks</name>
    <value>2</value>
    <description>
      This should be a prime number larger than multiple number of slave
      hosts, e.g. for 3 nodes set this to 17
    </description>
  </property>
</configuration>
```

```
</description>
</property>

<property>
<name>mapred.reduce.tasks</name>
<value>2</value>
<description>
This should be a prime number close to a low multiple of slave hosts,
e.g. for 3 nodes set this to 7
</description>
</property>

</configuration>
```

- File master chứa danh sách máy chủ, slaves chứa danh sách các máy slave
- Copy cài đặt trên ra tất cả các máy trong cụm

```
scp -r /hadoop-0.18.3/ machine2:/hadoop-0.18.3/
```

```
scp -r /hadoop-0.18.3/ machine3:/hadoop-0.18.3/
```

.....

- Khởi động hadoop

```
cd /nutch-0.9
```

```
bin/hadoop namenode -format
```

```
bin/start-all.sh
```

Có thể xem thông tin namenode và datanode ở <http://master:50070> như hình 9, xem thông tin jobtracker và tasktracker ở <http://master:50030> như hình 10.

Started: Thu May 20 16:03:29 ICT 2010
Version: 0.18.3, r736250
Compiled: Thu Jan 22 23:12:08 UTC 2009 by ndaley
Upgrades: There are no upgrades in progress.

[Browse the filesystem](#)

Cluster Summary

18 files and directories, 16 blocks = 34 total. Heap Size is 4.94 MB / 992.31 MB (0%)

Capacity : 137.66 GB
DFS Remaining : 95.52 GB
DFS Used : 10.02 MB
DFS Used% : 0.01 %
[Live Nodes](#) : 3
[Dead Nodes](#) : 0

Live Datanodes : 3

Node	Last Contact	Admin State	Size (GB)	Used (%)	Used (%)	Remaining (GB)	Blocks
master	1	In Service	39.15	0.01	<input type="text"/>	28.84	16
slave01	2	In Service	32	0	<input type="text"/>	10.03	5
slave02	2	In Service	66.51	0.01	<input type="text"/>	56.66	13

Hình 9: Giao diện namenode

Chương 5: Cài đặt thực nghiệm và đánh giá hiệu năng

State: RUNNING
Started: Thu May 20 16:04:49 ICT 2010
Version: 0.18.3, r736250
Compiled: Thu Jan 22 23:12:08 UTC 2009 by ndaley
Identifier: 201005201604

Cluster Summary

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg.
0	0	1	3	6	6	4.00

Running Jobs

Running Jobs
none

Completed Jobs

Completed Jobs							
Jobid	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Re
job_201005201604_0001	sislab	wordcount	100.00% <div><div></div></div>	4	4	100.00% <div><div></div></div>	1

Hình 10: Giao diện JobTracker

5.3. Chạy thử và đánh giá hiệu năng

Đánh giá hiệu năng của cụm Hadoop đã cài đặt trên Nutch bằng chương trình WordCounter.

Thư mục /test chứa 4 file txt, kích thước 4 file lần lượt là

1,139,024 bytes

396,147 bytes

674,762 bytes

1,573,044 bytes

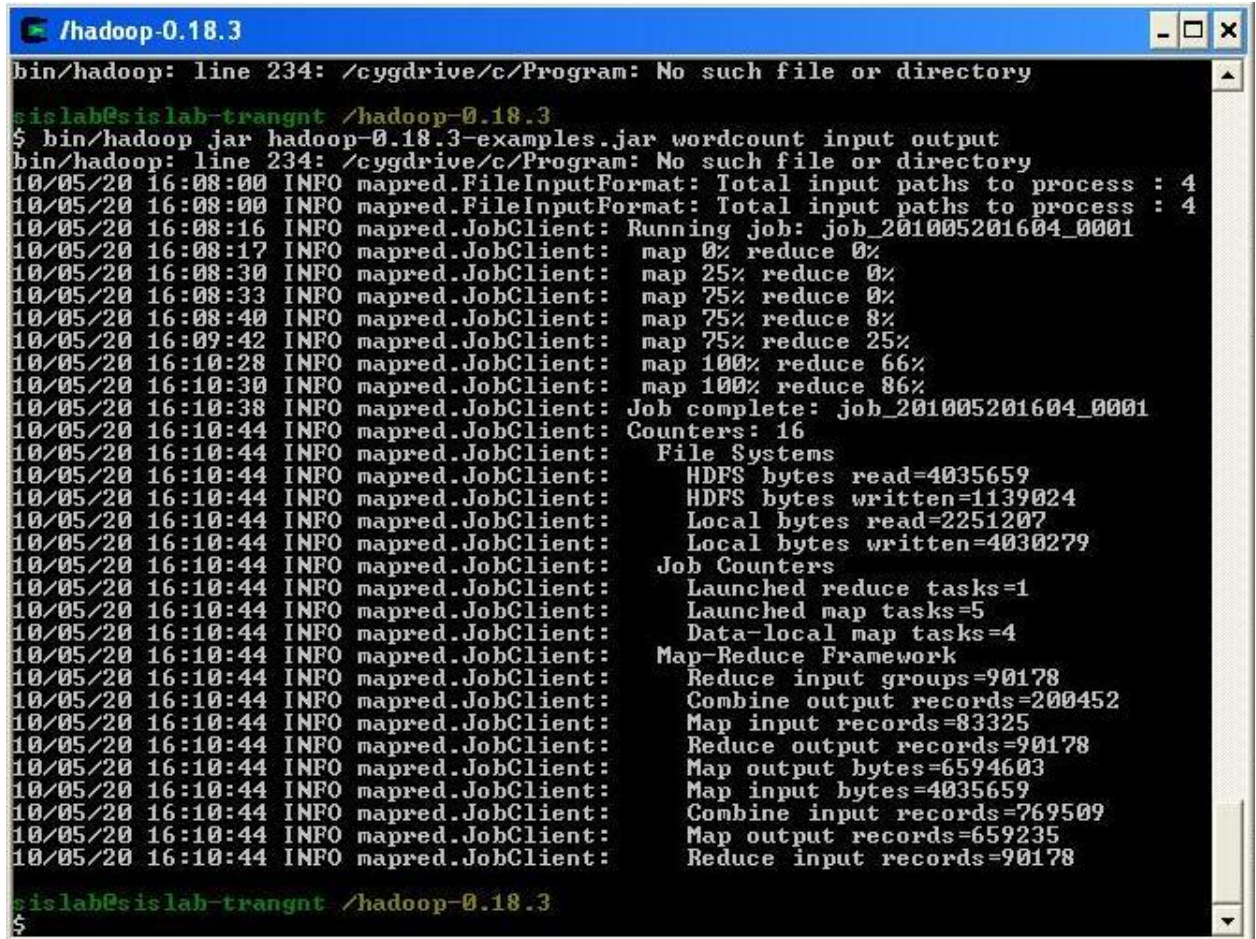
WordCounter sẽ đếm các từ có trong 4 file ở thư mục này

```
bin/hadoop dfs -put /test input
```

```
bin/hadoop dfs -ls
```

```
bin/hadoop dfs -ls input
```

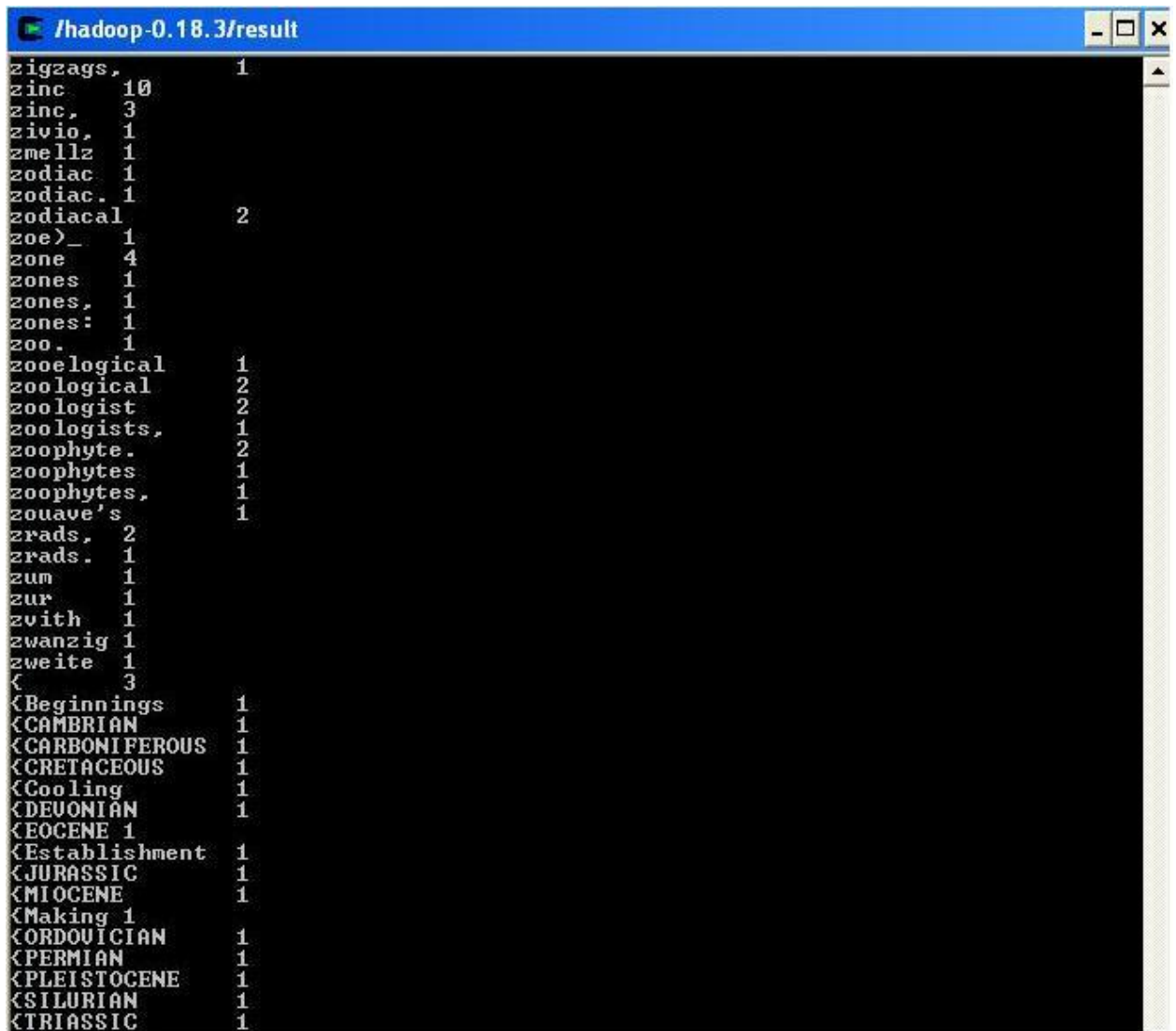
```
bin/hadoop jar hadoop-0.18.3.examples.jar wordcount input output
```



```
/hadoop-0.18.3
bin/hadoop: line 234: /cygdrive/c/Program: No such file or directory
sislab@sislab-trangnt /hadoop-0.18.3
$ bin/hadoop jar hadoop-0.18.3-examples.jar wordcount input output
bin/hadoop: line 234: /cygdrive/c/Program: No such file or directory
10/05/20 16:08:00 INFO mapred.FileInputFormat: Total input paths to process : 4
10/05/20 16:08:16 INFO mapred.JobClient: Running job: job_201005201604_0001
10/05/20 16:08:17 INFO mapred.JobClient: map 0% reduce 0%
10/05/20 16:08:30 INFO mapred.JobClient: map 25% reduce 0%
10/05/20 16:08:33 INFO mapred.JobClient: map 75% reduce 0%
10/05/20 16:08:40 INFO mapred.JobClient: map 75% reduce 8%
10/05/20 16:09:42 INFO mapred.JobClient: map 75% reduce 25%
10/05/20 16:10:28 INFO mapred.JobClient: map 100% reduce 66%
10/05/20 16:10:30 INFO mapred.JobClient: map 100% reduce 86%
10/05/20 16:10:38 INFO mapred.JobClient: Job complete: job_201005201604_0001
10/05/20 16:10:44 INFO mapred.JobClient: Counters: 16
10/05/20 16:10:44 INFO mapred.JobClient: File Systems
10/05/20 16:10:44 INFO mapred.JobClient: HDFS bytes read=4035659
10/05/20 16:10:44 INFO mapred.JobClient: HDFS bytes written=1139024
10/05/20 16:10:44 INFO mapred.JobClient: Local bytes read=2251207
10/05/20 16:10:44 INFO mapred.JobClient: Local bytes written=4030279
10/05/20 16:10:44 INFO mapred.JobClient: Job Counters
10/05/20 16:10:44 INFO mapred.JobClient: Launched reduce tasks=1
10/05/20 16:10:44 INFO mapred.JobClient: Launched map tasks=5
10/05/20 16:10:44 INFO mapred.JobClient: Data-local map tasks=4
10/05/20 16:10:44 INFO mapred.JobClient: Map-Reduce Framework
10/05/20 16:10:44 INFO mapred.JobClient: Reduce input groups=90178
10/05/20 16:10:44 INFO mapred.JobClient: Combine output records=200452
10/05/20 16:10:44 INFO mapred.JobClient: Map input records=83325
10/05/20 16:10:44 INFO mapred.JobClient: Reduce output records=90178
10/05/20 16:10:44 INFO mapred.JobClient: Map output bytes=6594603
10/05/20 16:10:44 INFO mapred.JobClient: Map input bytes=4035659
10/05/20 16:10:44 INFO mapred.JobClient: Combine input records=769509
10/05/20 16:10:44 INFO mapred.JobClient: Map output records=659235
10/05/20 16:10:44 INFO mapred.JobClient: Reduce input records=90178
sislab@sislab-trangnt /hadoop-0.18.3
$
```

Hình 11: Kết quả chạy ví dụ WordCount

Có thể đọc kết quả trực tiếp bằng lệnh `bin/hadoop dfs -cat output/*`



```
/hadoop-0.18.3/result
zigzags, 1
zinc 10
zinc, 3
zivio, 1
zmellz 1
zodiac 1
zodiac, 1
zodiacal 2
zoe)_ 1
zone 4
zones 1
zones, 1
zones: 1
zoo, 1
zoological 1
zoological 2
zoologist 2
zoologists, 1
zoophyte, 2
zoophytes 1
zoophytes, 1
zouave's 1
zrads, 2
zrads, 1
zum 1
zur 1
zwith 1
zwanzig 1
zweite 1
< 3
<Beginnings 1
<CAMBRIAN 1
<CARBONIFEROUS 1
<CRETACEOUS 1
<Cooling 1
<DEVONIAN 1
<EOCENE 1
<Establishment 1
<JURASSIC 1
<MIOCENE 1
<Making 1
<ORDOVICIAN 1
<PERMIAN 1
<PLEISTOCENE 1
<SILURIAN 1
<TRIASSIC 1
```

Hình 12: Kết quả file output

Kết luận

Sau một thời gian tiếp cận và tìm hiểu về hệ cơ sở dữ liệu phân tán. Khóa luận đã đạt được một số kết quả sau:

- Những khái niệm cơ bản về hệ phân tán
- Phân tích Bigtable – một hệ thống phân tán rất nổi tiếng của Google, có ứng dụng rộng rãi trong các sản phẩm của hãng này cũng như trong công nghệ máy tìm kiếm nói chung
- Đi sâu tìm hiểu cấu trúc và cách thức hoạt động của hệ phân tán Hadoop, đặc biệt là cơ chế Map-Reduce

Do giới hạn về thời gian cũng như kiến thức của tác giả, khóa luận không tránh khỏi những thiếu sót về nhiều mặt. Rất mong được sự thông cảm của bạn đọc.

Tài liệu tham khảo

Tiếng Việt

- [1]. M. Tamer Ozsü, Patrick Valduriez - **Nguyên lý các hệ cơ sở dữ liệu phân tán**. Nhà xuất bản thống kê 1999, biên dịch : Trần Đức Quang
- [2]. TS. Nguyễn Bá Tường. Lý thuyết cơ sở dữ liệu phân tán.

Tiếng Anh

- [3]. Bentley, J., L., and McIlroy, M., D. Data compression using long common strings. In Data Compression Conference (1999), pp. 287-295.
- [4]. Bloom, B., H. **Space/time trade-offs in hash coding with allowable errors**. CACM 13, 7 (1970), 422-426.
- [5]. Burrows, M., **The Chubby lock service for loosely coupled distributed systems**. In Proc, of the 7th OSDI (Nov, 2006).
- [6]. Chandar, T., Griesemer, R., and Redstone, J. **Paxos made live – An engineering perspective**. In Proc. of PODC (2007).
- [7]. Comer, D. **Ubiquitous B-tree**. Computing Surveys 11,2 (June, 1979) 121-137.
- [8]. Dean, J., and Ghemawat, S., **MapReduce: Simplified data processing on large clusters**. In Proc. Of the 6th OSDI (Dec. 2004), pp. 137-150.
- [9]. Dewitt, D. J., and Gray, J. **Parallel database systems: The future of high performance database systems**. CACM 35,6 (June 1992), 85-88.
- [10]. Dewitt, D., Katz, R., Olken, F., Sharipo, L., Stonebraker, M., and Wood, D. **Implementation techniques for main memory database systems**. In Proc, of SIGMOD (June 1984), pp, 1-8.
- [11]. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C.Hsieh, Deborah A.Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber **Bigtable : A Distributed Storage System for Structured Data**. OSDI'06:

Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006.

[12]. Gawlick, D., and Kinkade, D. **Varieties of concurrency control in IMS/VS fast path**. Database Engineering Bulletin 8, 2 (1985), 3-10.

[13]. Ghemawat, S., Gobioff, H., and Leung, S. – T. **The Google filesystem**. In Proc, of the 19th ACM SOSP (Dec. 2003), pp. 29-43.

[14]. McCarthy, J. Recursive funtions of symbolic expressions and their computation by machine. CACM 3,4 (Apr. 1960) , 184-185.

[15]. Pike, R., Dorward, S., Griesemer, R., and Quinlan, S. Interpreting the data: Parallel analysis with Sawzall. Scientific Programming Journal 13, 4 (2005), 227-298.

[16]. Salmen, D., Malyuta, T., Feters, R., and Antunes, R. **Cloud data structure Diagramming Techniques and Degisn Patterns**.

[17]. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. **The Google File System**. 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2003.

[18]. Zhao, Z., and Pjesivac-Grbovic, J. **MapReduce- The programming Model and Practice**.

[19] Hadoop wike: <http://en.wikipedia.org/wiki/Hadoop>

[20] Hadoop homepage: <http://hadoop.apache.org/>

[21] Nutch homepage: <http://nutch.apache.org/>

[22] Nutch wiki: <http://wiki.apache.org/nutch>