Apache Hadoop framework.

#L3h4ng(11/2018)

“Big Data đang trở thành một phần trọng yếu của mỗi công ty, và Hadoop là công nghệ cốt lõi cho việc lưu trữ và truy cập số lượng lớn dữ liệu”

Mục Lục

I. Giới thiệu về Apache hadoop framework

1. Khái niệm

Hadoop là một Apache framework mã nguồn mở được viết bằng java, cho phép xử lý phân tán (distributed processing) các tập dữ liệu lớn trên các cụm máy tính (clusters of computers) thông qua mô hình lập trình đơn giản. Hadoop được thiết kế để mở rộng quy mô từ một máy chủ đơn sang hàng ngàn máy tính khác có tính toán và lưu trữ cục bộ (local computation and storage).

2. Lịch sử

Nguồn gốc của Hadoop đến từ các bài viết File System Google được xuất bản vào tháng 10 năm 2003. Bài viết này là một bài nghiên cứu được sản sinh ra từ Google – MapReduce: Xử lý dữ liệu đơn giản trên các cluster lớn. Bắt đầu phát triển trên dự án Apache Nutch, nhưng đã được chuyển qua dự án con Hadoop mới trong tháng 1 năm 2006. Doug Cutting đã làm việc tại Yahoo! vào thời điểm đó, đặt tên Hadoop theo tên của con voi đồ chơi của con trai mình.

Hadoop 0.1.0 được phát hành vào tháng 4 năm 2006 và tiếp tục phát triển bởi nhiều người đóng góp đến dự án Apache Hadoop.

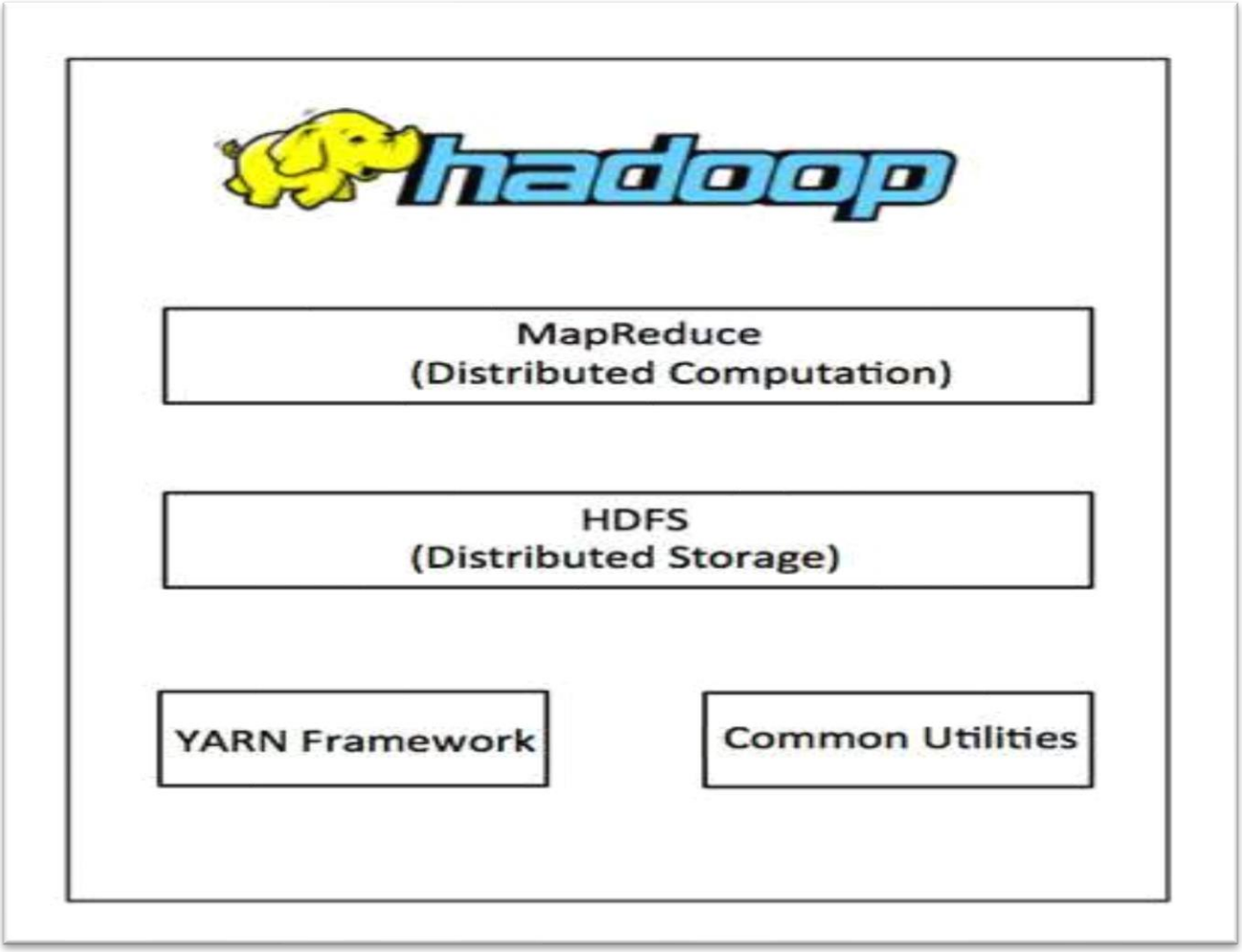
3. Kiến trúc của Hadoop framework

Hadoop framework gồm 4 module:

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

Có thể sử dụng sơ đồ sau để mô tả bốn thành phần có trong Hadoop framework.

Apache Hadoop framework. #L3h4ng(11/2018)



Kể từ 2012, thuật ngữ “Hadoop” không chỉ đề cập đến các module cơ sở nêu trên mà còn đề cập đến các gói phần mềm mở rộng (additional software packages) có thể được cài đặt bên cạnh Hadoop, chẳng hạn như Apache Pig, Apache Hive, Apache HBase, Apache Spark.

- **Hadoop Streaming:** Một tiện ích để tạo nên mã MapReduce bằng bất kỳ ngôn ngữ nào: C, Perl, Python, C++, Bash, v.v. Các ví dụ bao gồm một trình mapper Python và một trình reducer AWK.

- **Hive và Hue:** Nếu bạn thích SQL, bạn sẽ rất vui khi biết rằng bạn có thể viết SQL và yêu cầu Hive chuyển đổi nó thành một tác vụ MapReduce. Đúng là bạn chưa có một môi trường ANSI-SQL đầy đủ, nhưng bạn có 4000 ghi chép và khả năng mở rộng quy mô ra nhiều Petabyte. Hue cung cấp cho bạn một giao diện đồ họa dựa trên trình duyệt để làm công việc Hive của bạn.

- **Pig:** Một môi trường lập trình mức cao hơn để viết mã MapReduce. Ngôn ngữ Pig được gọi là Pig Latin. Bạn có thể thấy các quy ước đặt tên hơi khác thường

một chút, nhưng bạn sẽ có tỷ số giá-hiệu năng đáng kinh ngạc và tính sẵn sàng cao.

- **Sqoop:** Cung cấp việc truyền dữ liệu hai chiều giữa Hadoop và cơ sở dữ liệu quan hệ yêu thích của bạn.

- **Oozie:** Quản lý luồng công việc Hadoop. Oozie không thay thế trình lập lịch biểu hay công cụ BPM của bạn, nhưng nó cung cấp cấu trúc phân nhánh if-then-else và điều khiển trong phạm vi tác vụ Hadoop của bạn.

- **HBase:** Một kho lưu trữ key-value có thể mở rộng quy mô rất lớn. Nó hoạt động rất giống như một hash-map để lưu trữ lâu bền (với những người hâm mộ python, hãy nghĩ đến một từ điển). Nó không phải là một cơ sở dữ liệu quan hệ, mặc dù có tên là HBase.

- **FlumeNG:** Trình nạp thời gian thực để tạo luồng dữ liệu của bạn vào Hadoop. Nó lưu trữ dữ liệu trong HDFS và HBase. Bạn sẽ muốn bắt đầu với FlumeNG, để cải thiện luồng ban đầu.

- **Whirr:** Cung cấp Đám mây cho Hadoop. Bạn có thể khởi động một hệ thống chỉ trong vài phút với một tệp cấu hình rất ngắn.

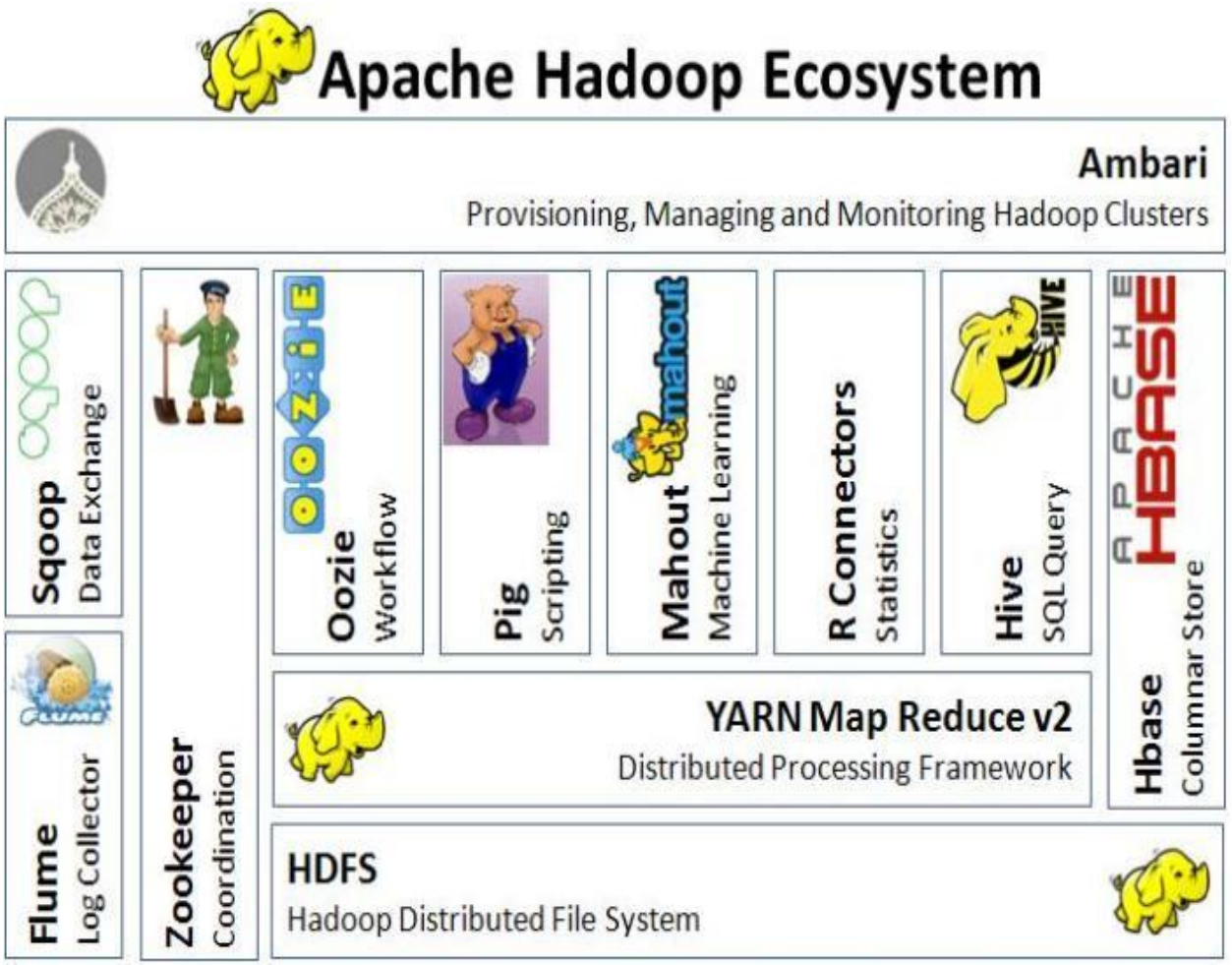
- **Mahout:** Máy học dành cho Hadoop. Được sử dụng cho các phân tích dự báo và phân tích nâng cao khác.

- **Fuse:** Làm cho hệ thống HDFS trông như một hệ thống tệp thông thường, do đó bạn có thể sử dụng lệnh ls, cd, rm và những lệnh khác với dữ liệu HDFS.

- **Zookeeper:** Được sử dụng để quản lý đồng bộ cho hệ thống. Bạn sẽ không phải làm việc nhiều với Zookeeper, nhưng nó sẽ làm việc rất nhiều cho bạn.

Các thành phần nói trên tạo nên hệ sinh thái hadopp:

Apac



Trong các thành phần nói trên, 2 thành phần quan trọng của Hadoop framework gồm:

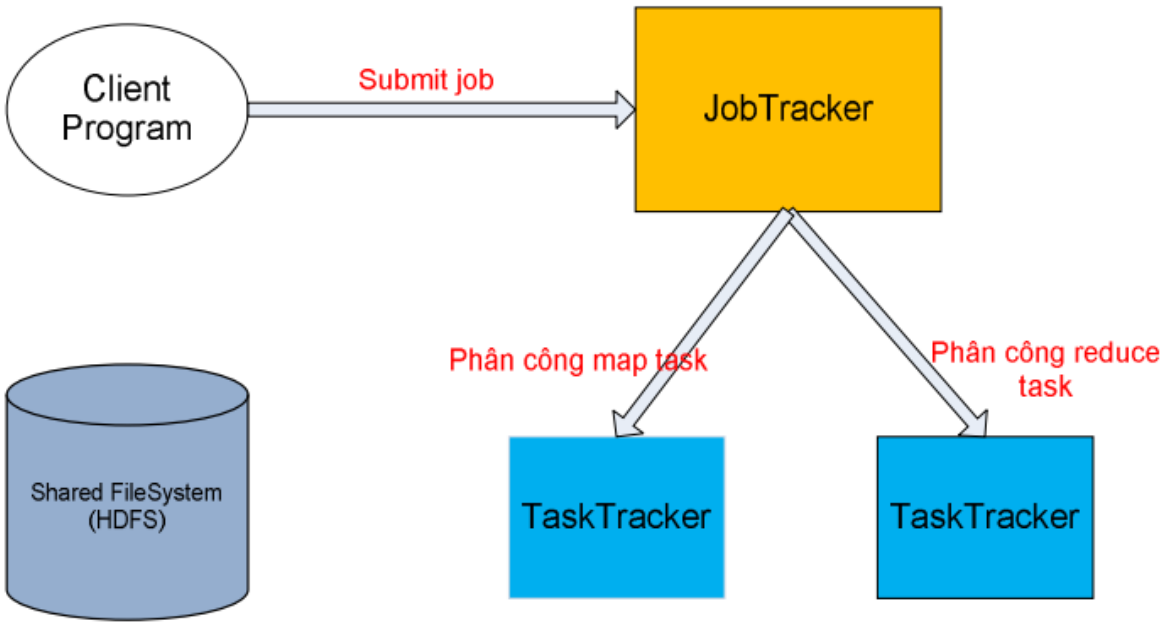
4. Kiến trúc MapReduce Engine

a. Các thành phần.

(Hình ảnh bên dưới)

Apache Hadoop framework.

#L3h4ng(11/2018)



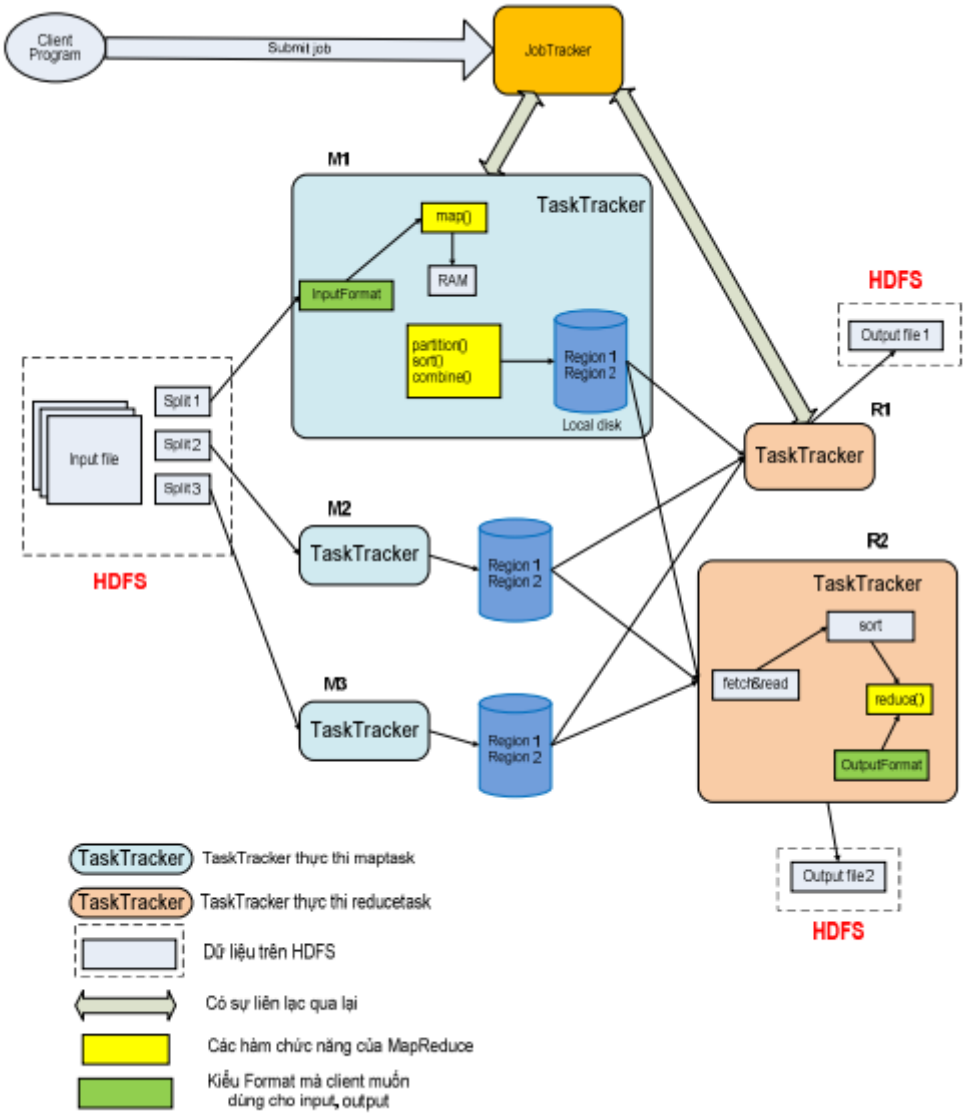
*** Client Program:** Chương trình HadoopMapReduce mà client đang sử dụng và tiến hành chạy một MapReduce Job.

*** JobTracker:** Tiếp nhận job và đảm nhận vai trò điều phối job này, nó có vai trò như bộ não của Hadoop MapReduce. Sau đó, nó chia nhỏ job thành các task, tiếp theo sẽ lên lịch phân công các task (map task, reduce task) này đến các tasktracker để thực hiện. Kèm theo vai trò của mình, JobTracker cũng có cấu trúc dữ liệu riêng của mình để sử dụng cho mục đích lưu trữ, ví dụ như nó sẽ lưu lại tiến độ tổng thể của từng job, lưu lại trang thái của các TaskTracker để thuận tiện cho thao tác lên lịch phân công task, lưu lại địa chỉ lưu trữ của các output của các TaskTracker thực hiện maptask trả về.

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

*** HDFS:** là hệ thống file phân tán được dùng cho việc chia sẻ các file dùng trong cả quá trình xử lý một job giữa các thành phần trên với nhau.

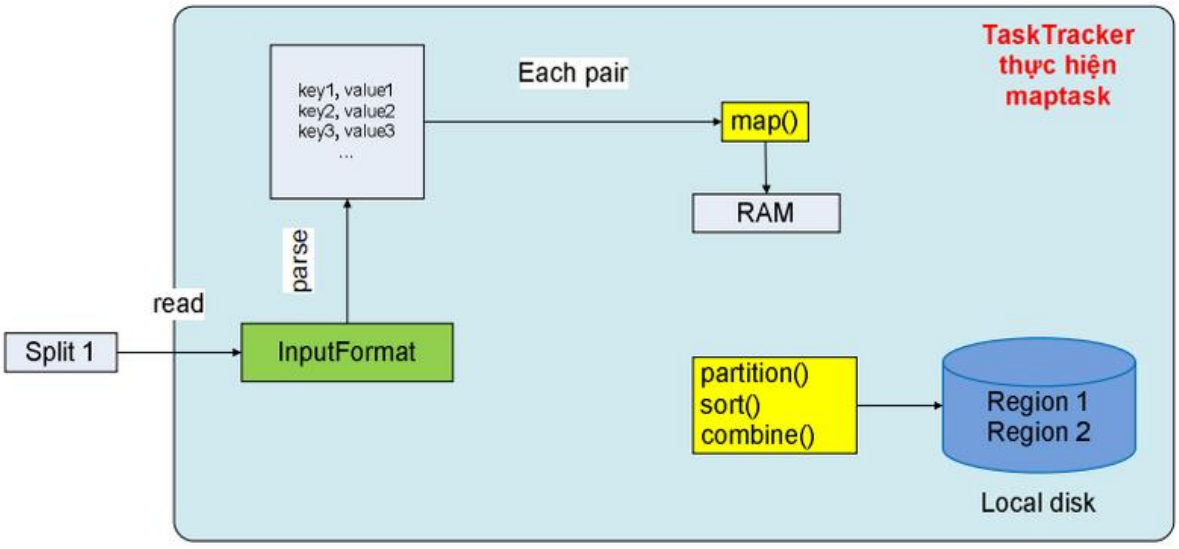
b) Cơ chế hoạt động (*Xem hình bên dưới*)



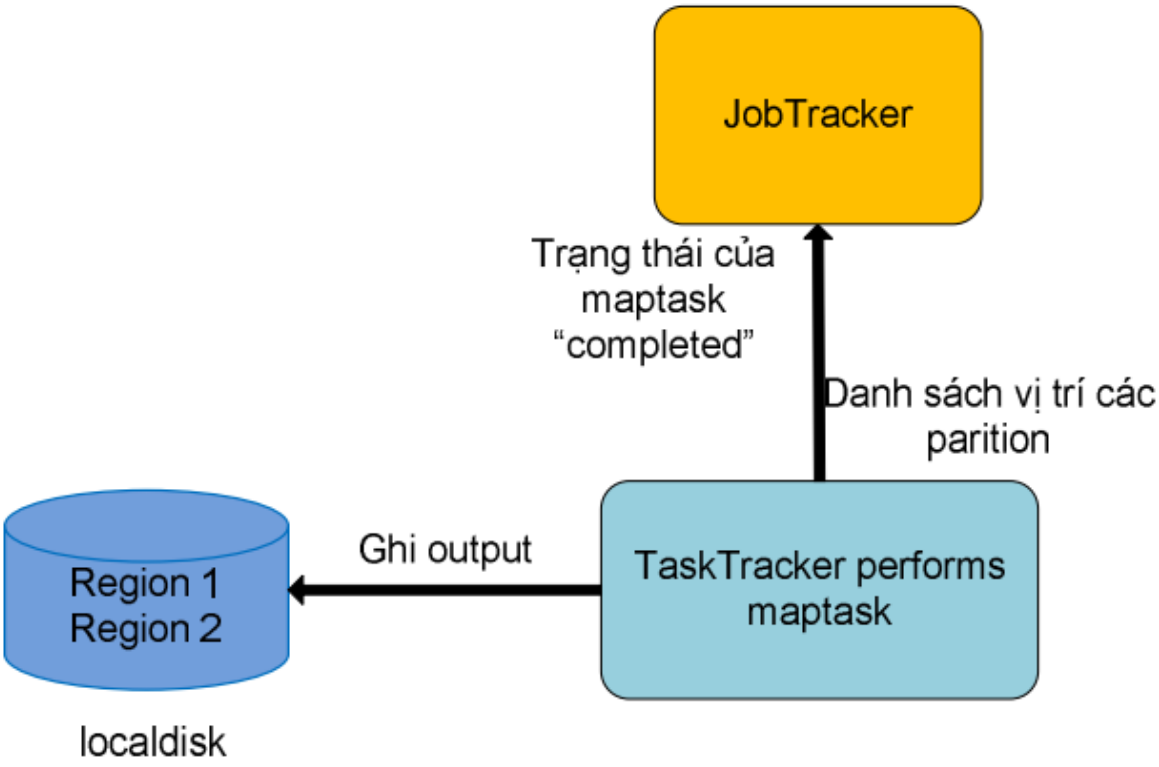
- Đầu tiên chương trình client sẽ yêu cầu thực hiện job và kèm theo là dữ liệu input tới JobTracker. JobTracker sau khi tiếp nhận job này, nó sẽ thông báo ngược về chương trình client tình trạng tiếp nhận job. Khi chương trình client nhận được thông báo nếu tình trạng tiếp nhận hợp lệ thì nó sẽ tiến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

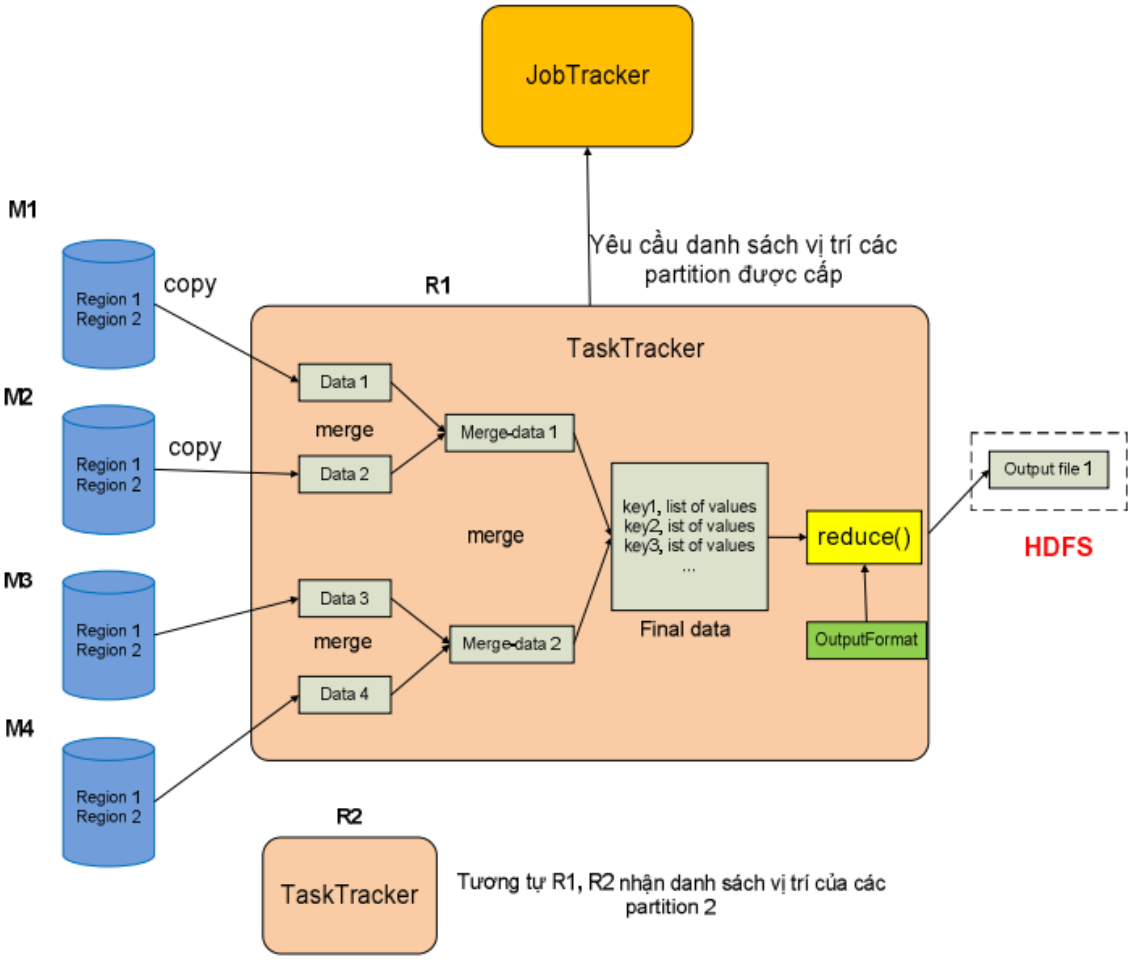
- Ngay khi JobTracker khởi tạo các thông tin cần thiết để chạy job, thì bên cạnh đó các TaskTracker trong hệ thống sẽ gửi các heartbeat đến JobTracker. Hadoop cung cấp cho các TaskTracker cơ chế gửi heartbeat đến JobTracker theo chu kỳ thời gian nào đó, thông tin bên trong heartbeat này cho phép JobTrack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



- 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



- 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



- 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.

5. 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 hân chia dữ liệu lên trên nhiều máy tính. Các hệ thống tập tin quản lý việc lưu trữ dữ liệu trên một mạng hiều máy tính gọi là hệ thống tập tin phân tán. Do hoạt động trên môi trường liên mạng, nên các hệ hống tập tin phân tán phức tạp hơn rất nhiều so với một hệ thống file cục bộ. Ví dụ như một hệ hống file phân tán

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

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

a. Giới thiệu

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

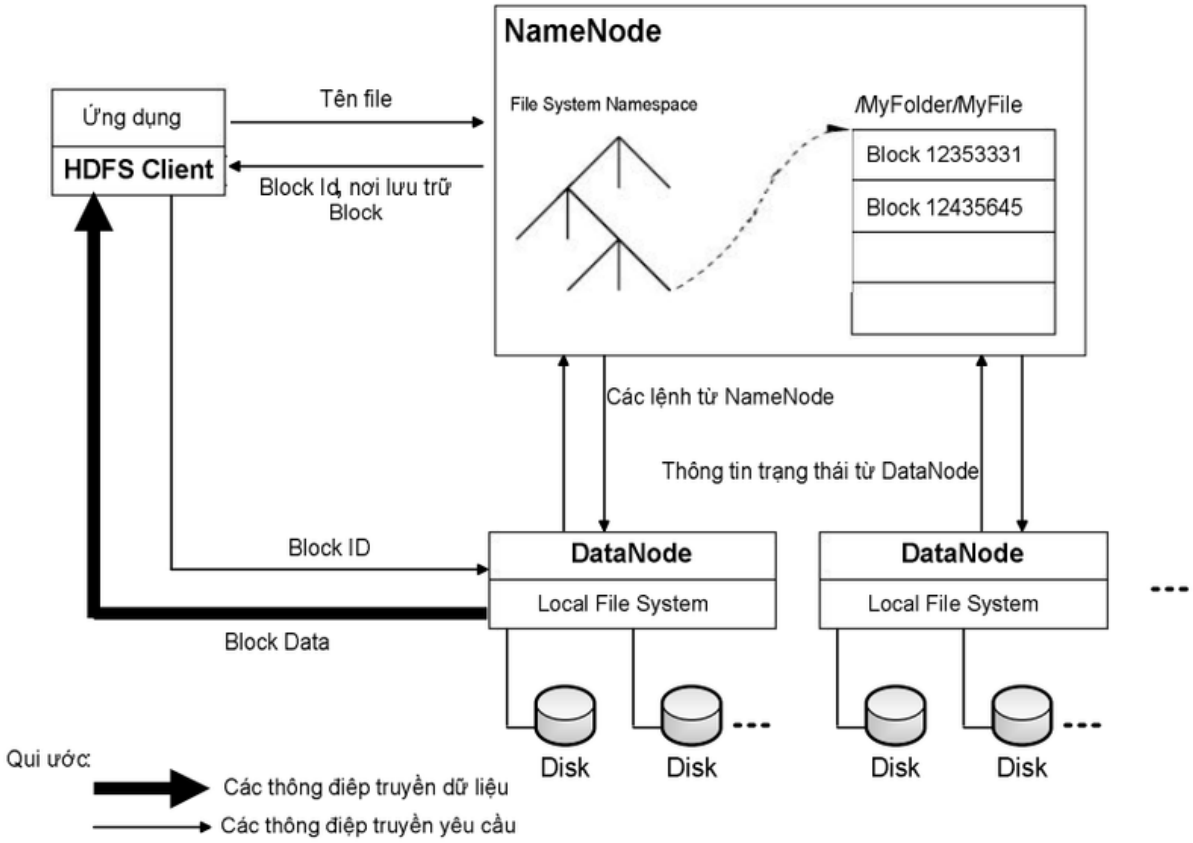
Thứ nhất, các lỗi về phần cứng sẽ thường xuyên xảy ra. Hệ thống HDFS sẽ chạy trên các cluster với hàng trăm hoặc thậm chí hàng nghìn node. Các node này được xây 18 dựng nên từ các phần cứng thông thường, giá rẻ, tỷ lệ lỗi cao. Chất lượng và số lượng của các thành phần phần cứng như vậy sẽ tất yếu dẫn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

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

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

b) Kiến trúc HDFS

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



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:

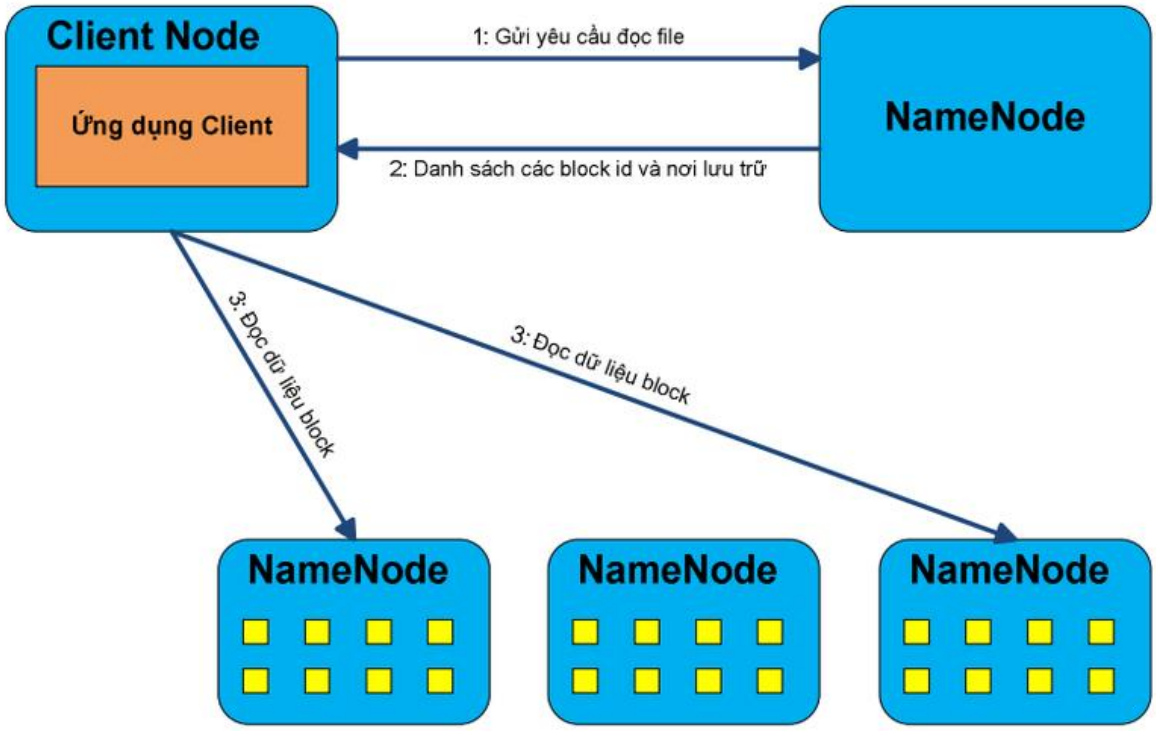
*** File System Namespace:** là hình ảnh cây thư mục của hệ thống file tại một thời điểm nào đó. File System namespace thể hiện tất các các file, thư mục có trên hệ thống file và quan hệ giữa chúng.

*** 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.

c) Quá trình đọc file trên HDFS

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



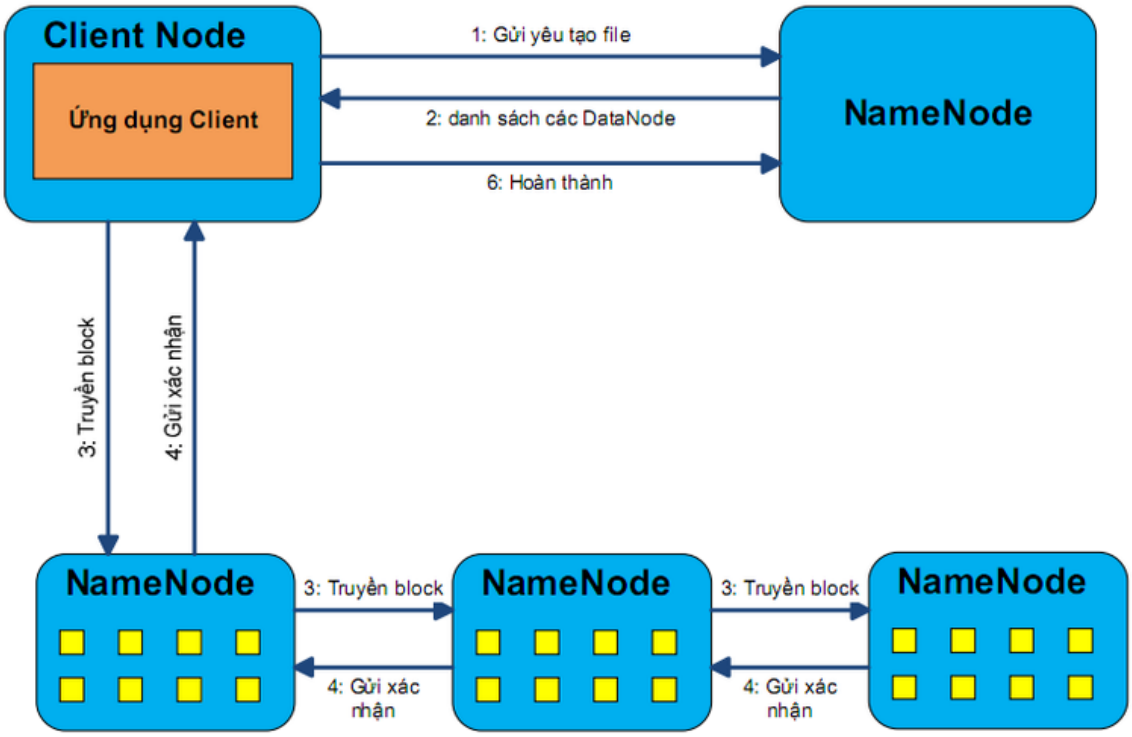
- Đầu tiên, client sẽ mở file cần đọc bằng cách gửi yêu cầu đọc file đến NameNode (1).Sau đó NameNode sẽ thực hiện một số kiểm tra xem file được yêu cầu đọc có tồn tại không, hoặc file cần đọc có đang ở trạng thái “khoẻ mạnh” hay không. Nếu mọi thứ đều ổn, NameNode sẽ gửi danh sách các block (đại diện 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.
- 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.

d) Ghi file trên HDFS

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.



- 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)
- Tiếp theo, client sẽ chia file cần ghi 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).
- Tiếp nữa, 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 được 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 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.

II. Hadoop hoạt động như thế nào?

1. Giải thích các thành phần

NameNode

Là một trình nền quan trọng nhất của Hadoop - các NameNode. Hadoop sử dụng một kiến trúc master/slave cho cả lưu trữ phân tán và xử lý phân tán. Hệ thống lưu trữ phân tán được gọi là Hadoop File System hay HDFS. NameNode là master của HDFS để chỉ đạo các trình nền DataNode slave để thực hiện các nhiệm vụ I/O mức thấp. NameNode theo dõi HDFS, cách các tập tin của bạn được phân chia thành các block, những node nào lưu các khối đó, và “kiểm tra sức khỏe” tổng thể của hệ thống tệp phân tán. Chức năng của NameNode là nhớ (memory) và I/O chuyên sâu. Như vậy, máy chủ lưu trữ NameNode thường không lưu trữ bất cứ dữ liệu người dùng hoặc

thực hiện bất cứ một tính toán nào cho một ứng dụng MapReduce để giảm khối lượng công việc trên máy. Điều này có nghĩa là máy chủ NameNode không gấp đôi (double) như là DataNode hay một TaskTracker.

Có điều đáng tiếc là có một khía cạnh tiêu cực đến tầm quan trọng của NameNode nó có một điểm của thất bại của một cụm Hadoop của bạn. Đối với bất cứ một trình nền khác, nếu các nút máy của chúng bị hỏng vì lý do phần mềm hay phần cứng, các Hadoop cluster có thể tiếp tục hoạt động thông suốt hoặc bạn có thể khởi động nó một cách nhanh chóng. Nhưng không thể áp dụng cho các NameNode.

DataNode

Mỗi máy slave trong cluster của bạn sẽ lưu trữ (host) một trình nền DataNode để thực hiện các công việc nào đó của hệ thống file phân tán - đọc và ghi các khối HDFS tới các file thực tế trên hệ thống file cục bộ (local filesytem). Khi bạn muốn đọc hay ghi một file HDFS, file đó được chia nhỏ thành các khối và NameNode sẽ nói cho các client của bạn nơi các khối trình nền DataNode sẽ nằm trong đó. Client của bạn liên lạc trực tiếp với các trình nền DataNode để xử lý các file cục bộ tương ứng với các block. Hơn nữa, một DataNode có thể giao tiếp với các DataNode khác để nhân bản các khối dữ liệu của nó để dự phòng.

Các DataNode thường xuyên báo cáo với các NameNode. Sau khi khởi tạo, mỗi DataNode thông báo với NameNode của các khối mà nó hiện đang lưu trữ. Sau khi Mapping hoàn thành, các DataNode tiếp tục thăm dò ý kiến NameNode để cung cấp thông tin về thay đổi cục bộ cũng như nhận được hướng dẫn để tạo, di chuyển hoặc xóa các blocks từ đĩa địa phương (local).

Secondary NameNode

Các Secondary NameNode (SNN) là một trình nền hỗ trợ giám sát trạng thái của các cụm HDFS. Giống như NameNode, mỗi cụm có một SNN, và nó thường trú trên một máy của mình. Không có các trình nền DataNode hay TaskTracker chạy trên cùng một server. SNN khác với NameNode trong quá trình xử lý của nó không nhận hoặc ghi lại bất cứ thay đổi thời gian thực tới HDFS. Thay vào đó, nó giao tiếp với các NameNode bằng cách chụp những bức ảnh của siêu dữ liệu HDFS (HDFS metadata) tại những khoảng xác định bởi cấu hình của các cluster.

Như đã đề cập trước đó, NameNode là một điểm truy cập duy nhất của lỗi (failure) cho một cụm Hadoop, và các bức ảnh chụp SNN giúp giảm thiểu thời gian ngừng (downtime) và mất dữ liệu. Tuy nhiên, một NameNode không đòi hỏi sự can thiệp của con người để cấu hình lại các cluster sẻ dùng SSN như là NameNode chính.

jobTracker

Trình nền JobTracker là một liên lạc giữa ứng dụng của bạn và Hadoop. Một khi bạn gửi mã nguồn của bạn tới các cụm (cluster), JobTracker sẽ quyết định kế hoạch thực hiện bằng cách xác định những tập tin nào sẽ xử lý, các nút được giao các nhiệm vụ khác nhau, và theo dõi tất cả các nhiệm vụ khi chúng đang chạy. Nếu một nhiệm vụ (task) thất bại (fail), JobTracker sẽ tự động chạy lại nhiệm vụ đó, có thể trên một node khác, cho đến một giới hạn nào đó được định sẵn của việc thử lại này.

Chỉ có một JobTracker trên một cụm Hadoop. Nó thường chạy trên một máy chủ như là một nút master của cluster.

TaskTraker

Như với các trình nền lưu trữ, các trình nền tính toán cũng phải tuân theo kiến trúc master/slave: JobTracker là giám sát tổng việc thực hiện chung của một công việc MapRecude và các taskTracker quản lý việc thực hiện các nhiệm vụ riêng trên mỗi node slave. Mỗi TaskTracker chịu trách nhiệm thực hiện các task riêng mà các JobTracker giao cho. Mặc dù có một TaskTracker duy nhất cho một node slave, mỗi TaskTracker có thể sinh ra nhiều JVM để xử lý các nhiệm vụ Map hoặc Reduce song song.

Một trong những trách nhiệm của các TaskTracker là liên tục liên lạc với JobTracker. Nếu JobTracker không nhận được nhịp đập từ một TaskTracker trong vòng một lượng thời gian đã quy định, nó sẽ cho rằng TaskTracker đã bị treo (cashed) và sẽ gửi lại nhiệm vụ tương ứng cho các nút khác trong cluster.

Cấu trúc liên kết này có một node Master là trình nền NameNode và JobTracker và một node đơn với SNN trong trường hợp node Master bị lỗi. Đối với các cụm nhở, thì SNN có thể thường chú trong một node slave. Mặt khác, đối với các cụm lớn, phân tách NameNode và JobTracker thành hai máy riêng. Các máy slave, mỗi máy chỉ lưu trữ một DataNode và Tasktracker, để chạy các nhiệm vụ trên cùng một node nơi lưu dữ liệu của chúng

2. Quá trình hoạt động

Giai đoạn 1

Một user hay một ứng dụng có thể submit một job lên Hadoop (hadoop job client) với yêu cầu xử lý cùng các thông tin cơ bản:

1. Nơi lưu (location) dữ liệu input, output trên hệ thống dữ liệu phân tán.
2. Các java class ở định dạng jar chứa các dòng lệnh thực thi các hàm map và reduce.
3. Các thiết lập cụ thể liên quan đến job thông qua các thông số truyền vào.

Giai đoạn 2

Hadoop job client submit job (file jar, file thực thi) và các thiết lập cho JobTracker. Sau đó, master sẽ phân phối tác vụ đến các máy slave để theo dõi và quản lý tiến trình các máy này, đồng thời cung cấp thông tin về tình trạng và chẩn đoán liên quan đến job-client.

Giai đoạn 3

TaskTrackers trên các node khác nhau thực thi tác vụ MapReduce và trả về kết quả output được lưu trong hệ thống file.

Khi “chạy Hadoop” có nghĩa là chạy một tập các trình nền - daemon, hoặc các chương trình thường trú, trên các máy chủ khác nhau trên mạng của bạn. Những trình nền có vai trò cụ thể, một số chỉ tồn tại trên một máy chủ, một số có thể tồn tại trên nhiều máy chủ.

Các daemon bao gồm:

- NameNode
- DataNode
- SecondaryNameNode
- JobTracker
- TaskTracker

3. Ưu điểm

- Hadoop framework cho phép người dùng nhanh chóng viết và kiểm tra các hệ thống phân tán. Đây là cách hiệu quả cho phép phân phối dữ liệu và công việc xuyên suốt các máy trạm nhờ vào cơ chế xử lý song song của các lõi CPU.
- Hadoop không dựa vào cơ chế chịu lỗi của phần cứng fault-tolerance and high availability (FTHA), thay vì vậy bản thân Hadoop có các thư viện được thiết kế để phát hiện và xử lý các lỗi ở lớp ứng dụng.
- Các server có thể được thêm vào hoặc gỡ bỏ từ cluster một cách linh hoạt và vẫn hoạt động mà không bị ngắt quãng.
- Một lợi thế lớn của Hadoop ngoài mã nguồn mở đó là khả năng tương thích trên tất cả các nền tảng do được phát triển trên Java.

_____*Hết*_____