Big Data

I. Mô hình lập trình MapReduce cho Bigdata

MapReduce là một kỹ thuật xử lý và là một mô hình lập trình cho tính toán phân tán để triển khai và xử lý dữ liệu lớn. MapReduce chứa 2 tác vụ quan trọng là map và reduce. WordCount là một ví dụ điển hình cho MapReduce mà sẽ được mình minh họa trong bài viết này

Tại sao Mapreduce lại ra đời?

Như phần giới thiệu thì ai cũng biết là MapReduce là viết gộp lại của map và reduce là 2 hàm chính trong phương pháp lập trình này. Mô hình lập trình này bắt nguồn chính là từ Google, hay rõ ràng hơn là từ 1 bài báo của Google. Vấn đề đặt ra là cần phải song song hóa các tính toán, phân tán dữ liệu và phải có khả năng chịu lỗi cao.

MapReduce đơn giản và mạnh mẽ cho phép tự động song song hóa và phân phối các phép tính quy mô lớn, kết hợp với việc triển khai mô hình lập trình này để đạt được hiệu suất cao trên các cụm máy tính lớn hàng trăm, hàng nghìn máy tính.

Mô hình lập trình

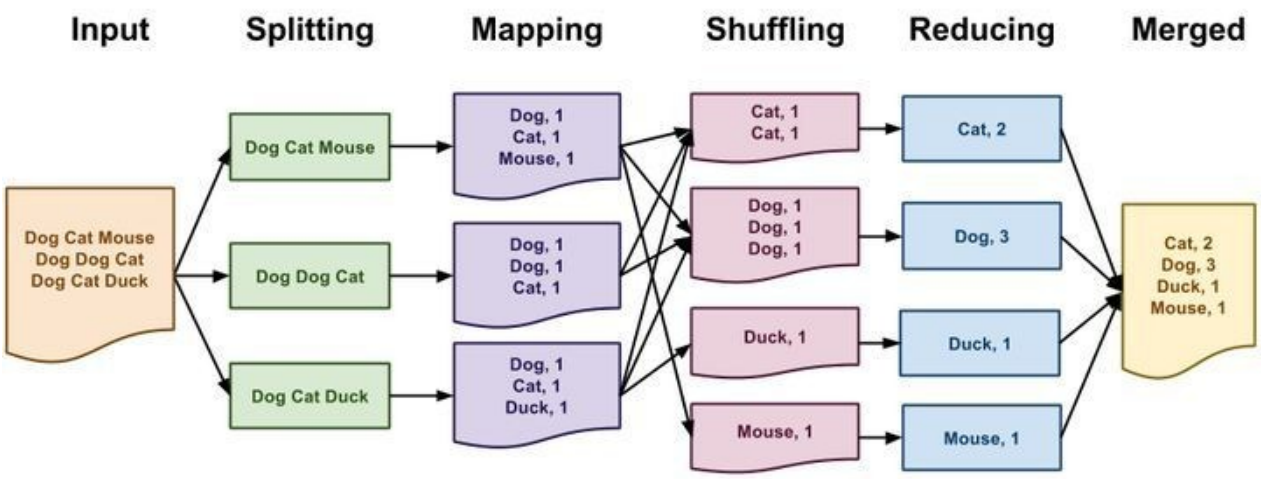
Mô hình của mapreduce phải nói là cực kì đơn giản và dễ hình dung. Lập trình viên sẽ chỉ phải viết lại 2 hàm trong mô hình lập trình MapReduce là hàm map và hàm reduce.

Ví dụ WordCount

WordCount là một bài toán kinh điển để minh họa cho MapReduce, ý tưởng của MapReduce được viết giống với đoạn mã giả sau:

- Ví dụ bây giờ chúng ta cần thực hiện 1 task là đếm số lần xuất hiện của mỗi từ trong 1 văn bản. Đầu tiên chúng ta sẽ chia tách văn bản đó thành các dòng, mỗi dòng sẽ được đánh 1 số thứ tự. Đầu vào của hàm Map sẽ là các cặp key/value chính là số thứ tự dòng / đoạn văn bản trên dòng đó.
- Trong Map chúng ta sẽ xử lý tách từng từ trong 1 dòng ra và gán cho chúng giá trị tần suất suất hiện ban đầu là 1.
- Sẽ có một tiến trình ở giữa giúp việc gộp các output đầu ra của Map có cùng key với nhau thành 1 mảng các giá trị.
- Đầu vào của Reduce sẽ là cặp key/value là từ / và 1 mảng là tần suất xuất hiện của từ đó. Trong Reduce chúng ta chỉ cần thực hiện cộng các giá trị trong mảng và đưa ra kết quả chính là số lần xuất hiện của từng từ trong văn bản đầu vào.

Để hình dung rõ hơn, bạn có thể xem hình ảnh sau:



Khi lập trình MapReduce các input, output cho 2 hàm Map và Reduce là thứ mà lập trình viên phải xác định trước khi làm. Đối với bài WordCount có vẻ đơn giản, nhưng trong các bài toán phức tạp hơn việc đưa về mô hình MapReduce chưa hẳn là đã dễ.

II. Giới thiệu tổng quan Hadoop

Hadoop là framework dựa trên 1 giải pháp tới từ Google để lưu trữ và xử lý dữ liệu lớn. Hadoop sử dụng giải thuật MapReduce xử lý song song các dữ liệu đầu vào. Tóm lại, Hadoop được sử dụng để phát triển các ứng dụng có thể thực hiện phân tích thống kê hoàn chỉnh trên dữ liệu số lượng lớn.

Kiến trúc Hadoop

Hadoop gồm 2 tầng chính:

- Tầng xử lý và tinh toán (MapReduce): MapReduce là một mô hình lập trình song song hóa để xử lý dữ liệu lớn trên một cụm gồm nhiều các máy tính thương mại (commodity hardware)
- Tầng lưu trữ (HDFS): HDFS cung cấp 1 giải pháp lưu trữ phân tán cũng được thiết kế để chạy trên các máy tính thương mại

Ngoài 2 thành phần được đề cập ở trên thì Hadoop framework cũng gồm 2 module sau:

- Hadoop Common: là các thư viện, tiện ích viết bằng ngôn ngữ Java
- Hadoop Yarn: lập lịch và quản lý các tài nguyên

Hadoop làm việc như thế nào?

Hadoop giải quyết vấn đề nếu như bạn hay công ty của bạn không đủ điều kiện để xây dựng các máy chủ siêu mạnh thì chúng ta có thể kết hợp nhiều các máy tính thương mại lại để mang lại một cụm máy có khả năng xử lý một lượng dữ liệu lớn. Các máy trong cụm có thể đọc và xử lý dữ liệu song song để mang lại những hiệu quả cao. Bạn có thể chạy code trên một cụm các máy tính, quá trình này thông qua luồng sau đây:

• Dữ liệu được phân chia vào các thư mục và file. Mỗi file được chứa trong 1 blocks có kích thước cố định được xác định sẵn (mặc định là 128MB)

• Các tệp này được phân phối trên các nút, cụm khác nhau

• HDFS nằm ở trên cùng của hệ thống file cục bộ, giám sát quá trình

• Các block được lưu các bản sao để đề phòng quá trình lỗi xảy ra trên phần cứng

• Kiểm tra mã được thực hiện thành công chưa

• Thực hiện bước Sort diễn ra giữa Map và Reduce

• Gửi data tới các máy nhất định để thực hiện các bước tiếp theo

• Viết log cho mỗi công việc hoàn thành

Ưu điểm của Hadoop

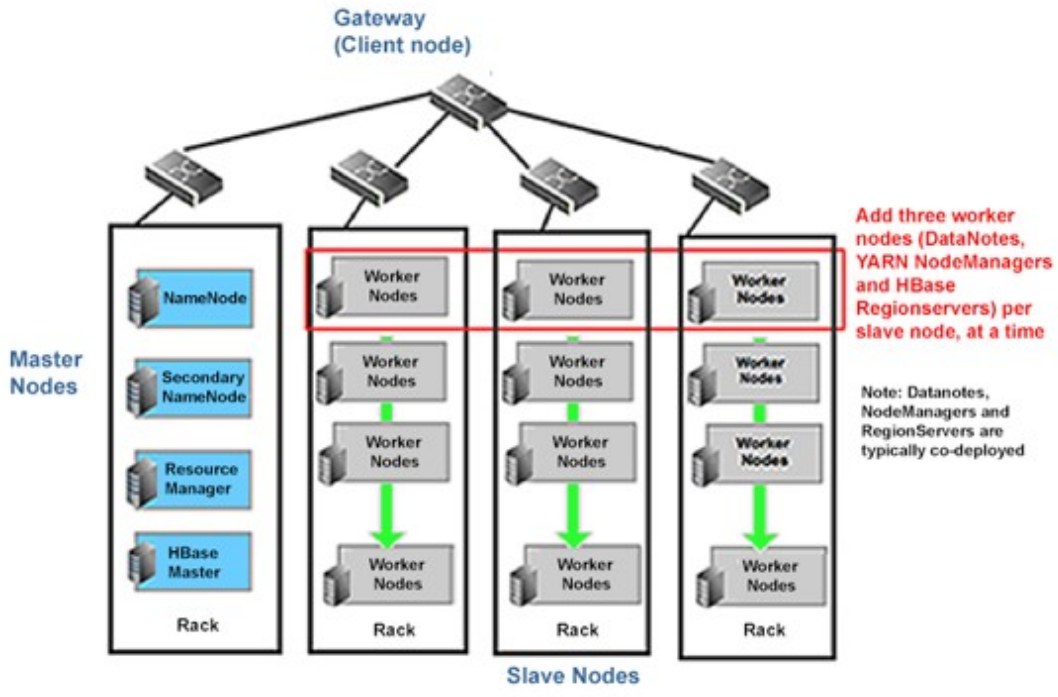
• Hadoop cho phép người dùng viết và kiểm tra nhanh trên hệ thống phân tán. Hadoop sử dụng hiệu quả và tự động phân tán dữ liệu và công việc qua nhiều máy trong cùng cụm.

• Hadoop không yêu cầu phần cứng của các máy trong cụm, bất cứ máy tính nào cũng có thể là 1 phần của cụm Hadoop. Hadoop sẽ phân công công việc hợp lý cho mỗi máy phù hợp với khả năng của mỗi máy.

• Hadoop cung cấp hệ thống có khả năng chịu lỗi và tính sẵn có cao. Thay vào đó thư viện Hadoop đã được thiết kế để xử lý lỗi từ tầng ứng dụng.

• Cụm có thể add thêm hoặc remove đi các cluster trong cụm mà không ảnh hưởng tới các tiến trình đang chạy

• Một lợi thế lớn khác của Hadoop là ngoài là mã nguồn mở, nó tương thích trên tất cả các nền tảng vì nó dựa trên Java



aaaa

III. Hadoop Ecosystem

Hệ sinh thái Apache Hadoop đề cập đến các thành phần khác nhau của thư viện phần mềm Apache Hadoop; nó bao gồm các dự án mã nguồn mở cũng như một loạt các công cụ bổ sung hoàn chỉnh khác. Một số công cụ nổi tiếng nhất của hệ sinh thái Hadoop bao gồm HDFS, Hive, Pig, YARN, MapReduce, Spark, HBase, Oozie, Sqoop, Zookeeper,…

Với HDFS, Hadoop MapReduce mình sẽ có các bài viết riêng sau, trong bài viết về Hadoop Ecosystem này chỉ mang tính chất liệt kê và giới thiệu các thành phần trong hệ sinh thái Hadoop.

HDFS

Hadoop Distributed File System (HDFS) là một trong những hệ thống lớn nhất trong hệ sinh thái Hadoop và là hệ thống lưu trữ chính của Hadoop.

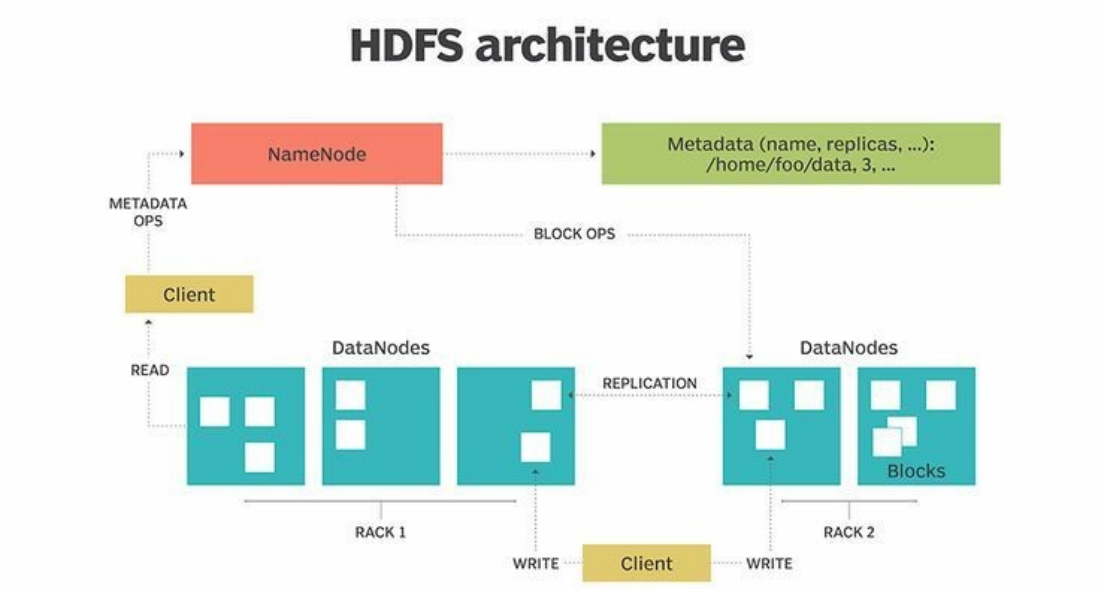
HDFS cung cấp khả năng lưu trữ tin cậy và chi phí hợp lí cho khối dữ liệu lớn, tối ưu cho các tập tin kích thước lớn (từ vài trăm MB cho tới vài TB). HDFS có không gian cây thư mục phân cấp giống như các hệ điều hành Unix, Linux.

Do các tính chất của dữ liệu lớn và hệ thống tập tin phân tán nên việc chỉnh sửa là rất khó khăn, Vì thế mà HDFS chỉ hỗ trợ việc ghi thêm dữ liệu vào cuối tệp (append), nếu bạn muốn chỉnh sửa ở bất kì chỗ khác chỉ có cách là viết lại toàn bộ tệp với các phần sửa đổi và thay thế lại tệp cũ. HDFS tuân theo tiêu chí “ghi một lần và đọc nhiều lần”.

Kiến trúc của HDFS là kiến trúc Master/Slave, HDFS master (namenode) quản lý không gian tên và các metadata, giám sát các datanode. HDFS slave (datanode) trực tiếp thao tác I/O với các chunks.

Nguyên lý thiết kế của HDFS là:

- Chỉ ghi thêm (append) => giảm chi phí điều khiển tương tranh
- Phân tán dữ liệu
- Nhân bản dữ liệu
- Cơ chế chịu lỗi



Hive

Apache Hive là một công cụ cơ sở hạ tầng kho dữ liệu để xử lý dữ liệu có cấu trúc trong Hadoop. Hive tạo điều kiện cho việc đọc, ghi và quản lý các tập dữ liệu lớn nằm trong bộ lưu trữ phân tán bằng cách sử dụng SQL (tuy nhiên hãy nhớ Hive không phải là một CSDL quan hệ).

Hive cung cấp ngôn ngữ kiểu SQL để truy vấn được gọi là HiveQL hoặc HQL.

Để tìm hiểu thêm về Hive bạn có thể xem thêm tại trang chủ của Hive: https://hive.apache.org/

HBase

HBase là một cơ sở dữ liệu dạng column-family, lưu trữ dữ liệu trên HDFS, được xem như là hệ quản trị CSDL của Hadoop.

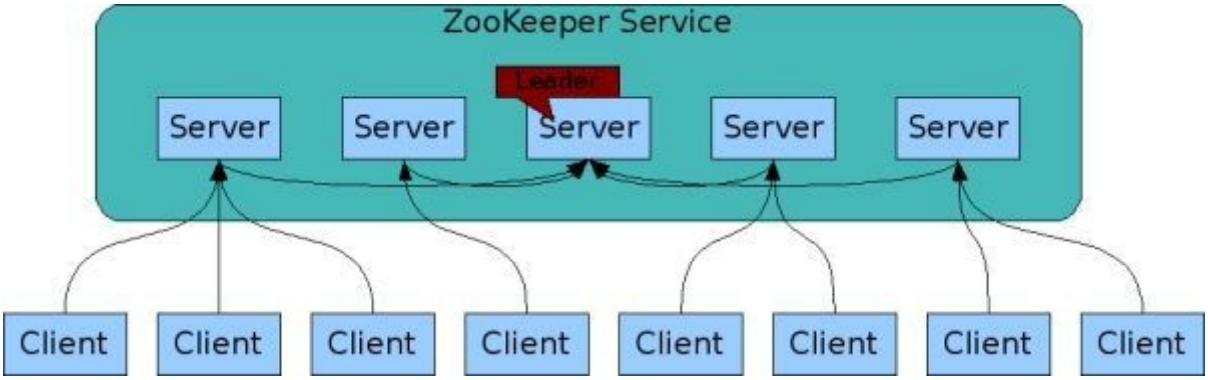
Để hiểu rõ hơn về Column-Family bạn có thể đọc thêm bài báo về Bigtable: Bigtable: A Distributed Storage System for Structured Data

Xem thêm về Apache HBase tại: https://hbase.apache.org/

Hadoop MapReduce

Đây là một lớp xử lý dữ liệu khác của Hadoop. Nó có khả năng xử lý dữ liệu có cấu trúc và phi cấu trúc lớn cũng như quản lý song song các tệp dữ liệu rất lớn bằng cách chia công việc thành một tập hợp các nhiệm vụ độc lập (sub-job).

Apache Zookeeper



Zookeeper là một dịch vụ cung cấp các chức năng phối hợp phân tán độ tin cậy cao:

- Quản lý các thành viên trong nhóm máy chủ
- Bầu cử leader
- Quản lý thông tin cấu hình động
- Giám sát trạng thái hệ thống

Đây là một dịch vụ lõi, tối quan trọng trong các hệ thống phân tán.

Xem thêm về Zookeeper tại https://zookeeper.apache.org/

YARN

Apache Hadoop YARN (Yet Another Resource Negotiator) được giới thiệu từ Hadoop 2.0 là một công nghệ hỗ trợ quản lý tài nguyên và lập lịch công việc trong Hadoop.

Chúng ta có thể thấy sự hiện diện của YARN chính là 2 daemons:

- Node Managers
- Resource Manager

Apache Kafka

Apache Kafka là một hệ thống được tạo ra bởi linkedin nhằm phục vụ cho việc xử lý dữ liệu theo luồng (stream process) sau đó được open-source. Ban đầu nó được nhìn nhận dưới dạng một message queue nhưng sau này được phát triển thành một nền tảng xử lý phân tán

IV. HDFS

Hadoop Distributed File System (HDFS) là hệ thống lưu trữ phân tán được thiết kế để chạy trên các phần cứng thông dụng. HDFS có khả năng chịu lỗi cao được triển khai sử dụng các phần cứng giá rẻ. HDFS cung cấp khả năng truy cập thông lượng cao vào dữ liệu ứng dụng vì thế nó rất phù hợp với ứng dụng có tập dữ liệu lớn.

Mục tiêu của HDFS

• Tiết kiệm chi phí cho việc lưu trữ dữ liệu lớn: có thể lưu trữ dữ liệu megabytes đến petabytes, ở dạng có cấu trúc hay không có cấu trúc.

• Dữ liệu có độ tin cậy cao và có khả năng khắc phục lỗi: Dữ liệu lưu trữ trong HDFS được nhân bản thành nhiều phiên bản và được lưu tại các DataNode khác nhau, khi có 1 máy bị lỗi thì vẫn còn dữ liệu được lưu tại DataNode khác.

• Tính chính xác cao: Dữ liệu lưu trữ trong HDFS thường xuyên được kiểm tra bằng mã checksum được tính trong quá trình ghi file, nếu có lỗi xảy ra sẽ được khôi phục bằng các bản sao.

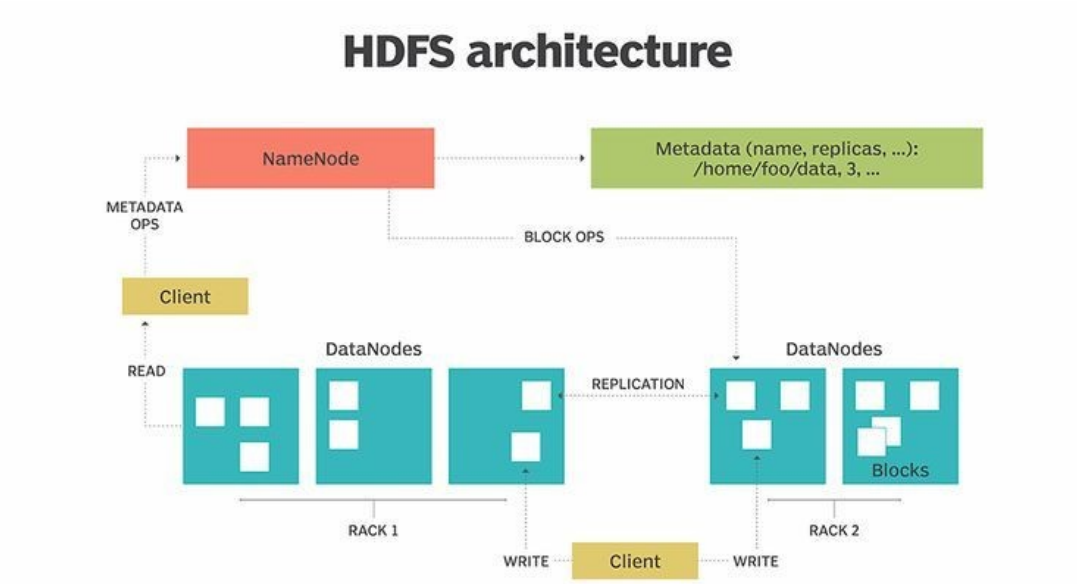
• Khả năng mở rộng: có thể tăng hàng trăm node trong một cluster.

• Có throughput cao: tốc độ xử lý truy nhập dữ liệu cao.

• Data Locality: xử lý dữ liệu tại chỗ.

HDFS Architecture

Theo dõi hình vẽ dưới để xem tổng quát về kiến trúc của HDFS.



Với HDFS, dữ liệu được ghi trên 1 máy chủ và có thể đọc lại nhiều lần sau đó tại bất cứ máy chủ khác trong cụm HDFS. HDFS bao gồm 1 Namenode chính và nhiều Datanode kết nối lại thành một cụm (cluster).

Namenode

HDFS chỉ bao gồm duy nhất 1 namenode được gọi là master node thực hiện các nhiệm vụ:

• Lưu trữ metadata của dữ liệu thực tế (tên, đường dẫn, blocks id, cấu hình datanode vị trí blocks,…)

• Quản lý không gian tên của hệ thống file (ánh xạ các file name với các blocks, ánh xạ các block vào các datanode)

• Quản lý cấu hình của cụm

• Chỉ định công việc cho datanode

Datanode

Chức năng của Datanode:

•Lưu trữ dữ liệu thực tế

•Trực tiếp thực hiện và xử lý công việc (đọc/ghi dữ liệu)

Secondary Namenode

Secondary Namenode là một node phụ chạy cùng với Namenode, nhìn tên gọi nhiều người nhầm tưởng rằng nó để backup cho Namenode tuy nhiên không phải vậy, Secondary Namenode như là một trợ lý đắc lực của Namenode, có vai trò và nhiệm vụ rõ ràng:

•Nó thường xuyên đọc các file, các metadata được lưu trên RAM của datanode và ghi vào ổ cứng.

•Nó liên đọc nội dung trong Editlogs và cập nhật vào FsImage, để chuẩn bị cho lần khởi động tiếp theo của namenode.

•Nó liên tục kiểm tra tính chính xác của các tệp tin lưu trên các datanode.

Cơ chế heartbeat

Heartbeat là cách liên lạc hay là cách để datanode cho namenode biết là nó còn sống. Định kì datanode sẽ gửi một heartbeat về cho namenode để namenode biết là datanode đó còn hoạt động. Nếu datanode không gửi heartbeat về cho namenode thì namenode coi rằng node đó đã hỏng và không thể thực hiện nhiệm vụ được giao. Namenode sẽ phân công task đó cho một datanode khác.

Rack

Theo thứ tự giảm dần từ cao xuống thấp thì ta có Rack > Node > Block. Rack là một cụm datanode cùng một đầu mạng, bao gồm các máy vật lí (tương đương một server hay 1 node) cùng kết nối chung 1 switch

Blocks

Blocks là một đơn vị lưu trữ của HDFS, các data được đưa vào HDFS sẽ được chia thành các block có các kích thước cố định (nếu không cấu hình thì mặc định nó là 128MB).

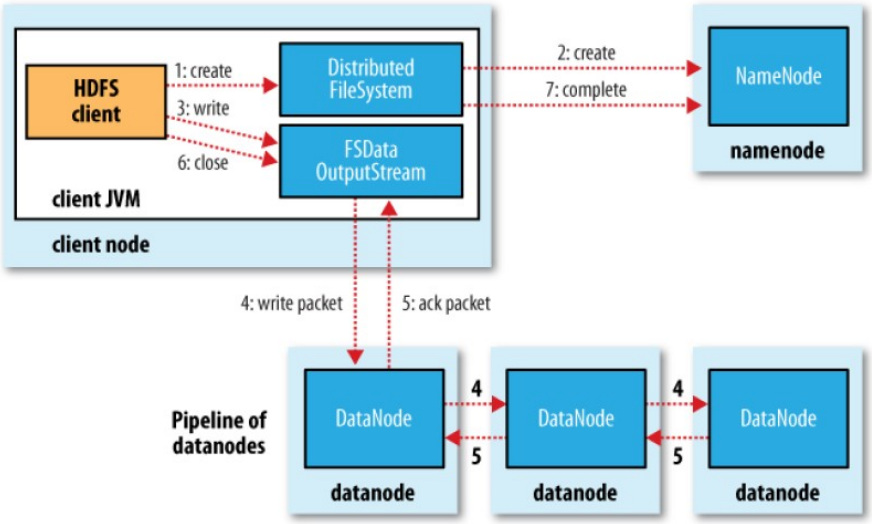
Vấn đề gì xảy ra nếu lưu trữ các file nhỏ trên HDFS?

Câu trả lời: HDFS sẽ không tốt khi xử lý một lượng lớn các file nhỏ. Mỗi dữ liệu lưu trữ trên HDFS được đại diện bằng 1 blocks với kích thước là 128MB, vậy nếu lưu trữ lượng lớn file nhỏ thì sẽ cần 1 lượng lớn các block để lưu chữ chúng và mỗi block chúng ta chỉ cần dùng tới 1 ít và còn thừa rất nhiều dung lượng gây ra sự lãng phí. Chúng ta cũng có thể thấy là block size của hệ thống file ở các hệ điều hành tiêu biểu như linux là 4KB là rất bé so với 128MB.

Hoạt động

Sau đây mình sẽ trình bày nguyên lí chung của đọc ghi dữ liệu trên HDFS:

Write data



Theo trình tự trong hình ta có các bước write dữ liệu như sau:

1. Client gửi yêu cầu tạo file ở DistributedFileSystem APIs.

2. DistributedFileSystem yêu cầu tạo file ở NameNode.NameNode kiểm tra quyền của client và kiểm tra file mới có tồn tại hay không…

3. DistributedFileSystem return FSDataOutPutStream cho client để ghi dữ liệu. FSDataOutputStream chứa DFSOutputStream, nó dùng để xử lý tương tác với NameNode và DataNode. Khi client ghi dữ liệu, DFSOutputStream chia dữ liệu thành các packet và đẩy nó vào hàng đợi DataQueue. DataStreamer sẽ nói với NameNode để phân bổ các block vào các datanode để lưu trữ các bản sao.

4. Các DataNode tạo thành pipeline, số datanode bằng số bản sao của file. DataStream gửi packet tới DataNode đầu tiên trong pipeline, datanode này sẽ chuyển tiếp packet lần lượt tới các Datanode trong pipeline.

5. DFSOutputStream có Ack Queue để duy trì các packet chưa được xác nhận bởi các DataNode. Packet ra khỏi ackqueue khi nhận được xác nhận từ tất cả các DataNode.

6. Client gọi close() để kết thúc ghi dữ liệu, các packet còn lại được đẩy vào pipeline.

7. Sau khi toàn bộ các packet được ghi vào các DataNode, thông báo hoàn thành ghi file.

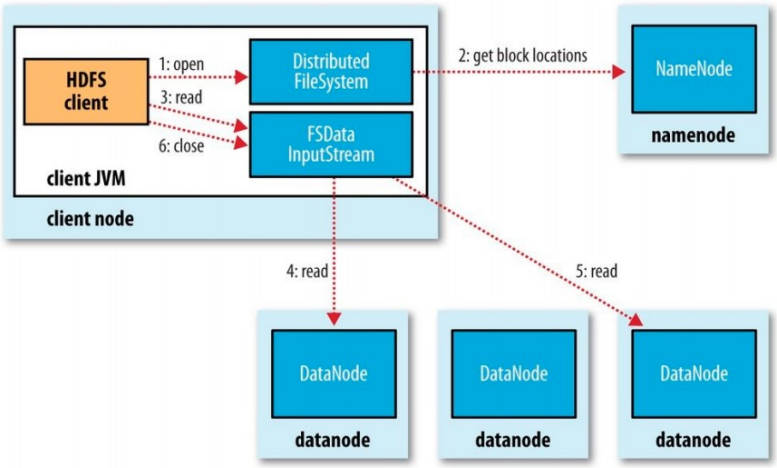
Đó là toàn bộ những gì diễn ra ở đằng sau, còn đây là ví dụ một trong những lênh thao tác ghi trên hdfs

```
hdfs dfs -put <path_on_your_computer> <path_on_hadoop>
```

Trong quá trình thực tế hầu như sẽ không làm việc trực tiếp với hệ thống file system của hadoop(HDFS) bằng câu lệnh, mà ta thường đọc, ghi qua spark, ví dụ

```
dataFrame.write.save("<path_on_hadoop>")
```

Read data



1. Để mở file, client gọi phương thức open ở FileSystemObject.

2.DistributedFileSystem gọi Name để lấy vị trí của blocks của file. NameNode trả về địa chỉ của các DataNode chứa bản sao của block đó.

3. Sau khi nhận được địa chỉ của các NameNode, Một đối tượng FSDataInputStream được trả về cho client. FSDataInputStream chứa DFSInputStream. DFSInputStream quản lý I/O của DataNode và NameNode.

4. Client gọi phương thức read() ở FSDataInputStream, DFSInputStream kết nối với DataNode gần nhất để đọc block đầu tiên của file. Phương thức read() được lặp đi lặp lại nhiều lần cho đến cuối block.

5. Sau khi đọc xong, DFSInputStream ngắt kết nối và xác định DataNode cho block tiếp theo. Khi DFSInputStream đọc file, nếu có lỗi xảy ra nó sẽ chuyển sang DataNode khác gần nhất có chứa block đó.

6. Khi client đọc xong file, gọi close().