HỆ THỐNG FILE LƯU TRỮ DỮ LIỆU PHÂN TÁN

Giảng viên: Nguyễn Tu Trung, Trần Mạnh Tuấn
BM HTTT, Khoa CNTT, Trường ĐH Thủy Lợi

Hà Nội, 2019

Nội dung

- Giới thiệu về HDFS
- Tổng quan thiết kế của HDFS
- Quá trình đọc file
- Quá trình ghi file
- Các tính năng của NameNode
- Khả năng chịu lỗi và chẩn đoán lỗi của HDFS

Giới thiệu về HDFS

- Khi kích thước của tập dữ liệu vượt quá khả năng lưu trữ của một máy tính => nhu cầu phân chia dữ liệu lên trên nhiều máy tính
- Các hệ thống tập tin quản lý việc lưu trữ dữ liệu trên một mạng nhiều máy tính gọi là hệ thống tập tin phân tán
- Do hoạt động trên môi trường liên mạng, nên các hệ thống tập tin phân tán phức tạp hơn rất nhiều so với một hệ thống file cục bộ
  - Ví dụ: một hệ thống file phân tán phải quản lý được tình trạng hoạt động (live/dead) của các server tham gia vào hệ thống file (để sử dụng hoặc loại trừ phần dữ liệu lưu trên server đó khi cần thiết)

Giới thiệu về HDFS

- 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
- Ra đời trên nhu cầu lưu trữ dữ liệu của Nutch, một dự án Search Engine nguồn mở
- Đã có rất nhiều Hadoop cluster chạy HDFS trên thế giới
  - Nổi bật nhất là của Yahoo với một cluster lên đến 1100 node với dung lượng HDFS 12 PB
  - Các công ty khác như Facebook, Adode, Amazon cũng đã xây dựng các cluster chạy HDFS với dung lượng hàng trăm, hàng nghìn TB

Giới thiệu về 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
- Cải thiện và bổ sung thêm các tính năng mới
  - Có khả năng phát hiện lỗi, chống chịu lỗi và tự động phục hồi: HDFS chạy trên các cluster với hàng trăm hoặc thậm chí hàng nghìn node, là các phần cứng thông thường, giá rẻ, tỷ lệ lỗi cao => tỷ lệ xảy ra lỗi trên cluster sẽ cao: 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… => HDFS đều có thể tự khắc phục để cho kết quả tin cậy
  - Làm việc với những file kích thước rất lớn bằng cách chia nhỏ thành các khối với kích thước tối ưu để hỗ trợ truy suất hiệu quả

Tổng quan thiết kế của HDFS

- Một số giả định trước khi thiết kế hệ thống HDFS
  - HDFS phải tự động phát hiện, khắc phục, và phục hồi kịp lúc khi các thành phần phần cứng bị hư hỏng vì hệ thống được xây dựng trên các phần cứng giá rẻ với khả năng hỏng hóc cao
  - Hệ thống lưu trữ và quản lý hiệu quả các tập tin có kích thước lớn (lên tới nhiều GB)
  - HDFS không phải là hệ thống file dành cho các mục đích chung, mà cho các ứng dụng dạng xử lý khối (batch processing) => Các file trên HDFS một khi được tạo ra, ghi dữ liệu và đóng lại thì không thể bị chỉnh sửa để đơn giản hoá, đảm bảo tính nhất quán của dữ liệu, cho phép truy cập dữ liệu với thông lượng cao

Tổng quan thiết kế của 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
- Mỗi file sẽ được chia ra làm một hay nhiều block và mỗi block có một block ID để nhận diện
- Các block của cùng một file (trừ block cuối cùng) có cùng kích thước và kích thước này được gọi là block size của file đó. Block size mặc định là 64MB, thường đặt là 128MB
- Mỗi block của file được lưu trữ thành nhiều bản sao khác nhau vì mục đích an toàn dữ liệu
- Mỗi cluster có một NameNode và một/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

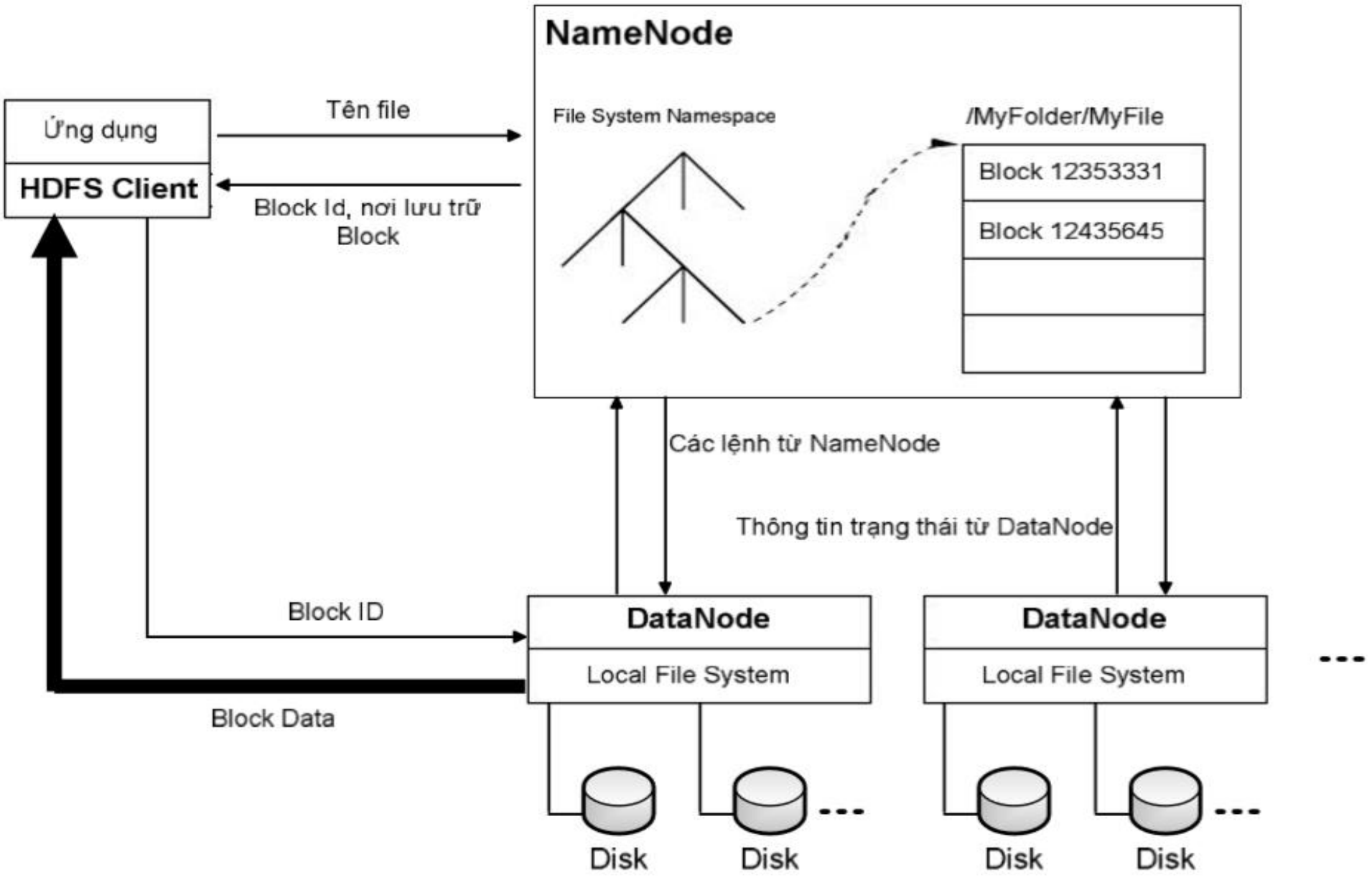
Tổng quan thiết kế của HDFS

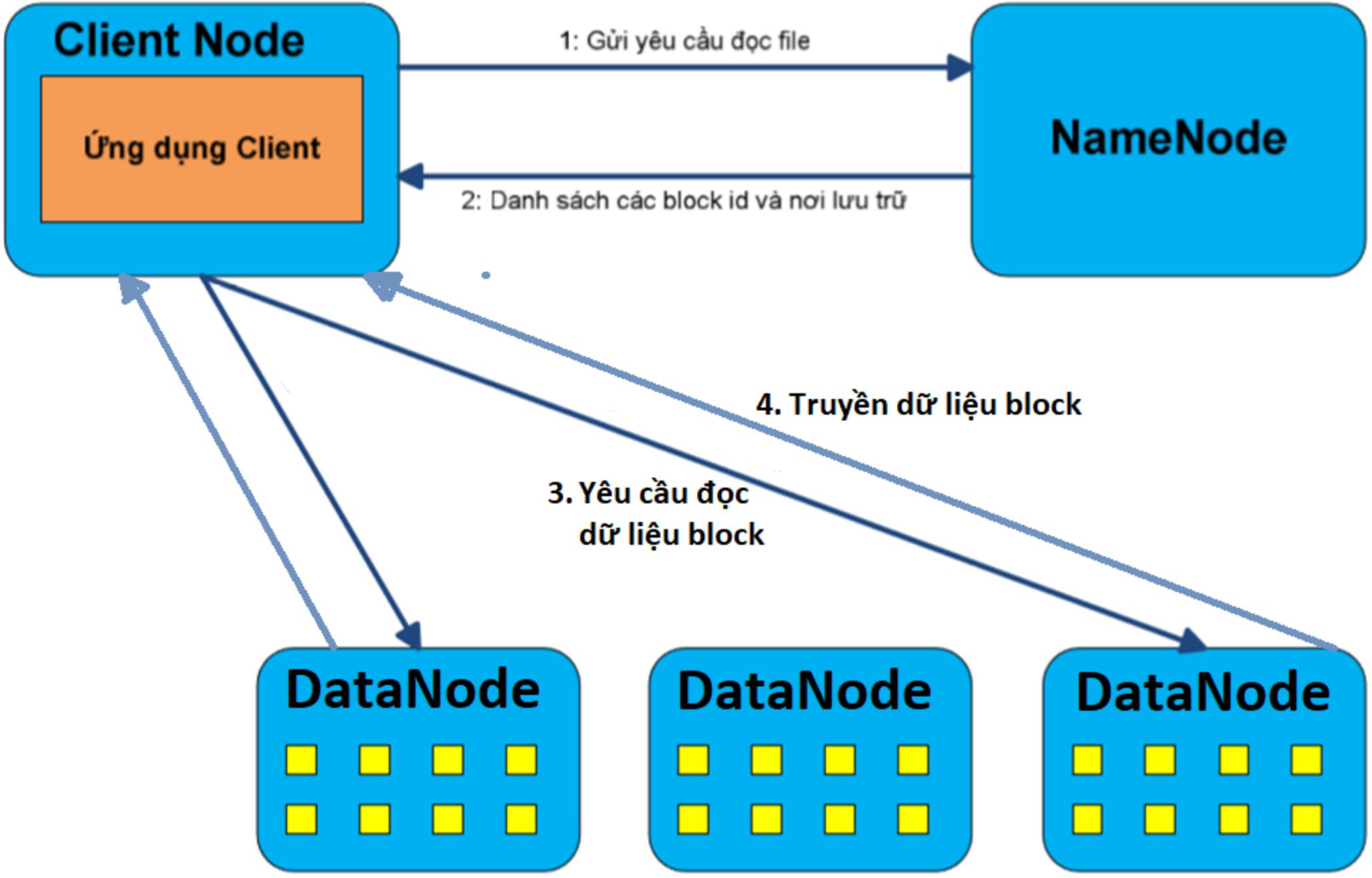
- Metadata mà Namenode lưu trữ gồm có:
  - File System Namespace: là hình ảnh cây thư mục của hệ thống file tại một thời điểm nào đó, thể hiện tất các các file, thư mục có trên hệ thống file và quan hệ giữa chúng
  - Thông tin để ánh xạ từ tên file ra thành danh sách các block: với mỗi file, ta có một danh sách có thứ tự các block (đại diện bởi Block ID) của file đó
  - Thông tin nơi lưu trữ các block: Mỗi block ta có một danh sách các DataNode lưu trữ các bản sao của block đó
- Datanode
  - Chịu trách nhiệm lưu trữ các block thật sự của từng file của hệ thống file phân tán lên hệ thống file cục bộ của Datanode
  - Mỗi 1 block được lưu trữ như là 1 file riêng biệt trên hệ thống file cục bộ của DataNode

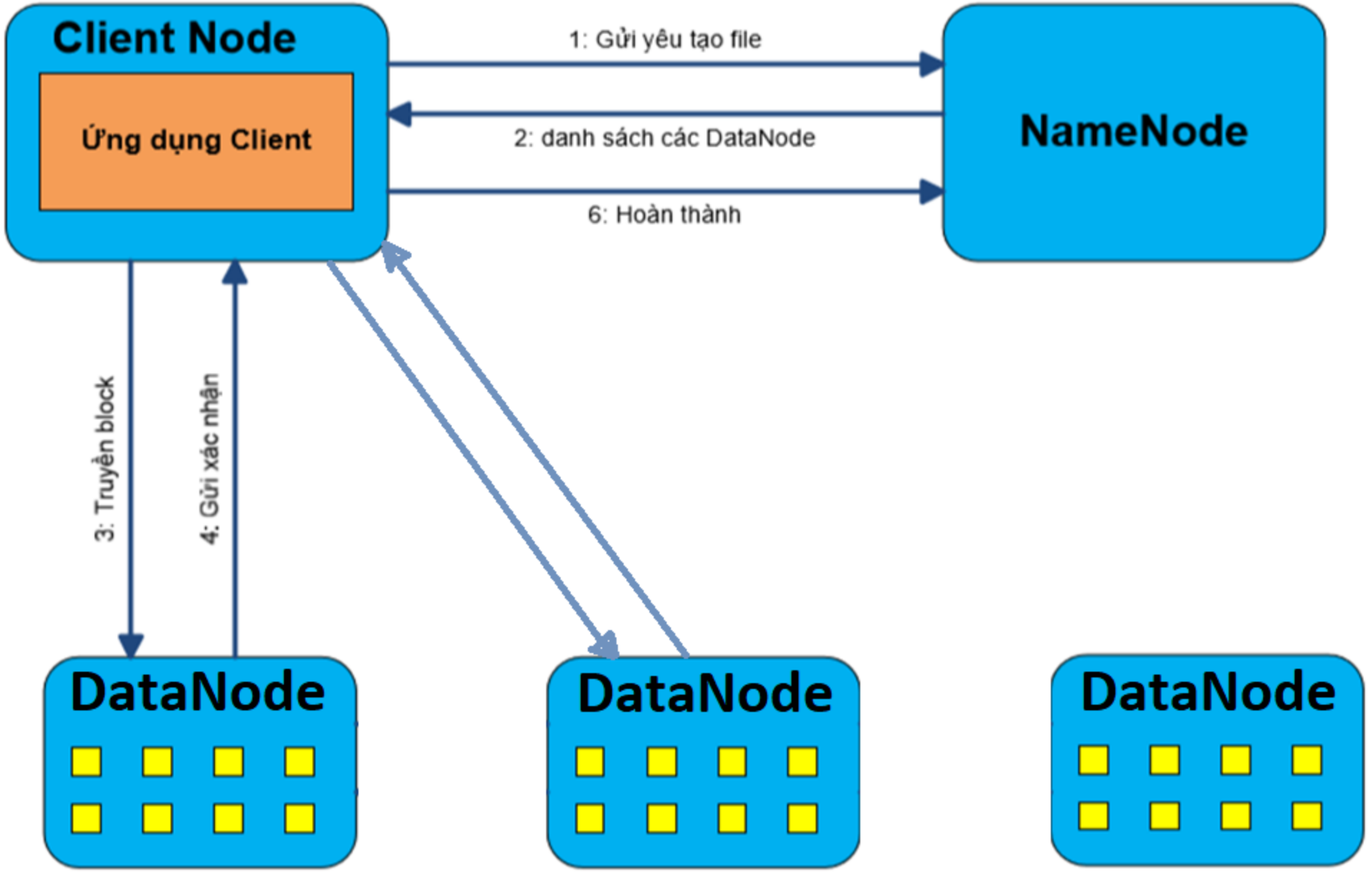
Tổng quan thiết kế của HDFS

- Khi client của hệ thống muốn đọc 1 file trên hệ thống HDFS
  - Client này sẽ thực hiện một request đến Namenode để lấy các metadata của file cần đọc => biết được danh sách các block của file và vị trí của các Datanode chứa các bản sao của từng block
  - Client kết nối trực tiếp với các Datanode để thực hiện các request đọc dữ liệu các block
- Định kỳ, mỗi DataNode báo cáo cho NameNode biết về danh sách tất cả các block mà nó đang lưu trữ
- Namenode dựa vào những thông tin này để cập nhật lại các Metadata trong nó
- Metadata trên namenode sẽ đạt được tình trạng thống nhất với dữ liệu trên các Datanode
- Metadata ở trạng thái thống nhất được dùng để nhân bản metadata dùng cho mục đích phục hồi lại NameNode nếu NameNode bị lỗi

Tổng quan thiết kế của HDFS




Client Node
Ứng dụng Client
1: Gửi yêu cầu đọc file
2: Danh sách các block id và nơi lưu trữ
NameNode
4. Truyền dữ liệu block
3. Yêu cầu đọc
dữ liệu block
DataNode
DataNode
DataNode


Client Node
Ứng dụng Client
1: Gửi yêu tạo file
2: danh sách các DataNode
6: Hoàn thành
NameNode
3: Truyền block
4: Gửi xác nhận
DataNode
DataNode
DataNode

Các tính năng của NameNode

- Nhận biết cấu trúc của cụm Hadoop
- Sắp xếp bản sao của các block lên các DataNode
- Cân bằng cluster
- Quản lý rác

Nhận biết cấu trúc của cụm Hadoop

- Mức cụm
  - Gồm bao nhiêu rack ?
  - Gồm những rack nào ?
- Mỗi rack
  - Gồm bao nhiêu node ?
  - Gồm những node nào ?
- Rack: Tập các node cùng switch

Sắp xếp bản sao của các block lên các DataNode

- Trên HDFS, một file được chia ra thành nhiều block
  - Mỗi block được lưu trữ thành N bản sao trên N DataNode khác nhau
  - N gọi là chỉ số mức độ sao chép (replication level)
  - Mỗi file có thể quy định một chỉ số replication level khác nhau
  - Chỉ số này càng cao thì file càng "an toàn"
- Nếu ghi tất cả bản sao cùng một node => Băng thông ghi tối ưu nhất, độ tin cậy thấp vì node có thể bị hỏng
- Nếu rải đều các node trong cụm thì => Băng thông ghi kém nhưng độ tin cậy cao
- 2 node thuộc cùng rack có khả năng cùng hỏng cao hơn thuộc 2 rack khác nhau (node hỏng hoặc switch hỏng)
- => Đảm bảo sự cân bằng giữa ba yếu tố: độ tin cậy, băng thông đọc và ghi dữ liệu: Các bản sao thường lưu trên 2 rack

Cân bằng cluster

- Theo thời gian sự phân bố của các block dữ liệu trên các DataNode có thể trở nên mất cân đối: một số node lưu trữ quá nhiều block dữ liệu trong khi một số node khác lại ít hơn
- Một cluster mất cân bằng có thể
  - Ảnh hưởng tới sự tối ưu hoá MapReduce
  - Tạo áp lực lên các DataNode lưu trữ quá nhiều block dữ liệu (lưu lượng truy cập từ client, dung lượng lưu trữ lớn)
- Một chương trình tên balancer (chạy như là một daemon trên NameNode) sẽ thực hiện việc cân bằng lại cluster
- Balancer định kỳ thực hiện phân tán lại các bản sao của block dữ liệu:
  - Bằng cách di chuyển nó từ các DataNode đã quá tải sang những DataNode còn trống
  - Vẫn đảm bảo các chiến lược sắp xếp bản sao của các block lên các DataNode

Quản lý rác

- Một file trên HDFS, sau khi bị xóa bởi người dùng hoặc ứng dụng, nó sẽ không lập tức bị xoá bỏ khỏi HDFS
- Trước hết, HDFS di chuyển file bị xóa đến thư mục rác có tên /trash
- Các tập tin sẽ được phục hồi nhanh chóng nếu như nó vẫn còn ở trong thư mục rác
- Sau một thời hạn nhất định (có thể cấu hình lại thời hạn này), NameNode sẽ thực sự xoá file trong thư mục rác này đi
- Việc xoá file kèm theo việc các bản sao của các block thuộc file đó sẽ thực sự bị xoá đi trên các DataNode
- Người dùng có thể lấy lại tập tin bị xoá bằng cách vào thư mục /trash và di chuyển nó ra, miễn là nó vẫn chưa thực sự bị xoá khỏi /trash

Khả năng chịu lỗi của HDFS

- Khả năng phục hồi nhanh chóng
- Nhân bản các block
- Nhân bản metadata trên NameNode với SecondaryNameNode

Khả năng phục hồi nhanh chóng

- NameNode và DataNode đều được thiết kế để có thể phục hồi nhanh chóng
- NameNode và DataNode liên lạc thông qua HeartBeat
  - HeartBeat là tín hiệu được gửi bởi DataNode đến NameNode sau khoảng thời gian thông thường để biểu thị sự hiện diện của nó (tức là nó còn sống)
  - Nếu sau khoảng thời gian nhất định, NameNode không nhận được bất kỳ phản hồi nào từ DataNode, thì DataNode đó đã ngưng hoạt động

Khả năng phục hồi nhanh chóng

- Trường hợp NameNode ngừng hoạt động
  - Chỉ cần phục hồi lại NameNode mà không cần phải restart tất cả các DataNode
  - NameNode sau khi phục hồi sẽ tự động liên lạc lại với các DataNode => hệ thống lại phục hồi (thực chất là NameNode chỉ đứng yên và lắng nghe các HeartBeat từ các DataNode)
- Nếu một DataNode bất kỳ bị ngừng hoạt động
  - Chỉ cần khởi động lại DataNode này
  - DataNode sẽ tự động liên lạc với NameNode thông qua các HeartBeat để cập nhật lại tình trạng của mình trên NameNode

Nhân bản các block

- Mỗi block dữ liệu trên HDFS được lưu trữ trùng lặp trên các DataNode khác nhau thuộc các rack khác nhau
- Người dùng (hoặc ứng dụng) có thể gán các chỉ số mức độ nhân bản (replication level) khác nhau cho các file khác nhau, tuỳ vào mức độ quan trọng của file đó, chỉ số mặc định là 3
- Khi một hay một số DataNode bị ngừng hoạt động, ta vẫn còn ít nhất một bản sao của block

Nhân bản metadata trên NameNode với SecondaryNameNode

- Từ kiến trúc cụm Hadoop, ta thấy tầm quan trọng của NameNode: lưu giữ tất cả các metadata của hệ thống
- Nếu Namenode gặp phải sự cố gì đó (cả phần cứng hay phần mềm) thì tất cả các file trên hệ thống HDFS đều sẽ bị mất, vì ta không có cách nào để tái cấu trúc lại các file từ các block được lưu trên các DataNode
- => Lý do có sự tồn tại của SecondaryNamenode
- SecondaryNamenode là một node duy nhất trên Hadoop cluster
- Nhiệm vụ của SecondaryNamenode là lưu trữ lại checkpoint (trạng thái thống nhất của metadata) mới nhất trên NameNode
- Khi NameNode gặp sự cố, checkpoint mới nhất này được import vào NameNode => NameNode trở lại hoạt động như thời điểm SecondaryNamenode tạo checkpoint
- SecondaryNamenode thực hiện nhiệm vụ của nó thông qua một daemon tên secondarynamenode