



GV: TS. Hồ Bảo Quốc

Nhóm HV: Đinh Thị Lương 1011036

Đoàn Cao Nghĩa 1011043

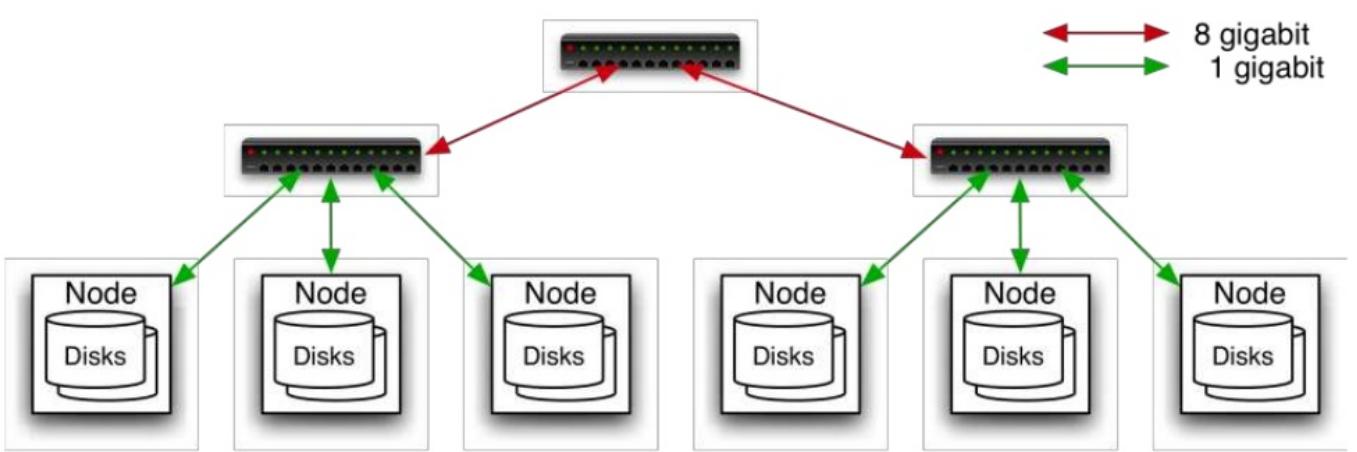
Hồng Xuân Viên 1011067

Nội dung

- Giới thiệu
 - Nhu cầu thực tế
 - Hadoop là gì?
 - Lịch sử phát triển
- Các thành phần của Hadoop
 - Hadoop Common, HDFS, MapReduce

Nhu cầu thực tế

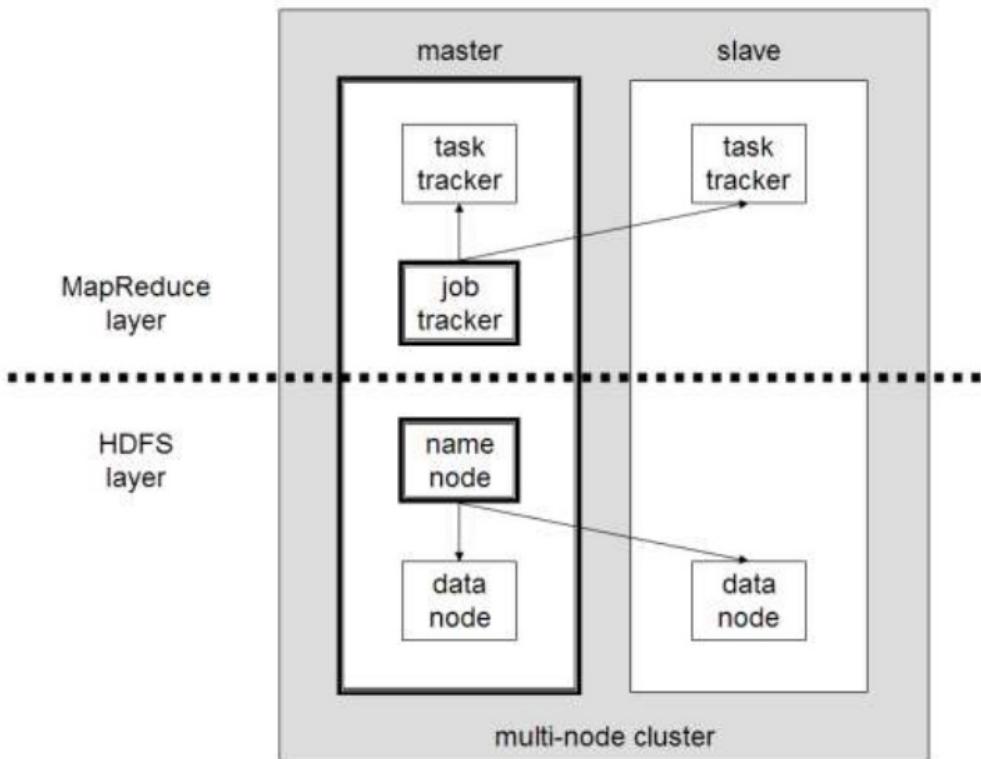
- Nhu cầu lưu trữ & xử lý dữ liệu hàng exabyte (1 exabyte = 10^{21} bytes)
 - Đọc & truyền tải dữ liệu rất chậm
- Cần rất nhiều node lưu trữ với chi phí thấp
 - Lỗi phần cứng ở node xảy ra hàng ngày
 - Kích thước cluster không cố định
- Nhu cần cần có một hạ tầng chung
 - Hiệu quả, tin cậy



- Kiến trúc 2 tầng
- Các node là các PC
- Chia làm nhiều rack (khoảng 40 PC/rack)

Hadoop là gì?

- Nền tảng ứng dụng hỗ trợ các ứng dụng phân tán với dữ liệu rất lớn.
 - Quy mô: hàng terabyte dữ liệu, hàng ngàn node.
- Thành phần:
 - Lưu trữ: HDFS (Hadoop Distributed Filesystem)
 - Xử lý: MapReduce
 - ✓ Hỗ trợ mô hình lập trình Map/Reduce



Lịch sử phát triển

- **2002-2004:** Doug Cutting giới thiệu Nutch
- **12/2004** – công bố bài báo về GFS & MapReduce
- **05/2005** – Nutch sử dụng MapReduce & DFS
- **02/2006** – Trở thành subproject của Lucene
- **04/2007** – Yahoo chạy 1000-node cluster
- **01/2008** – trở thành dự án cao cấp của Apache
- **07/2008** – Yahoo thử nghiệm 4000 node cluster

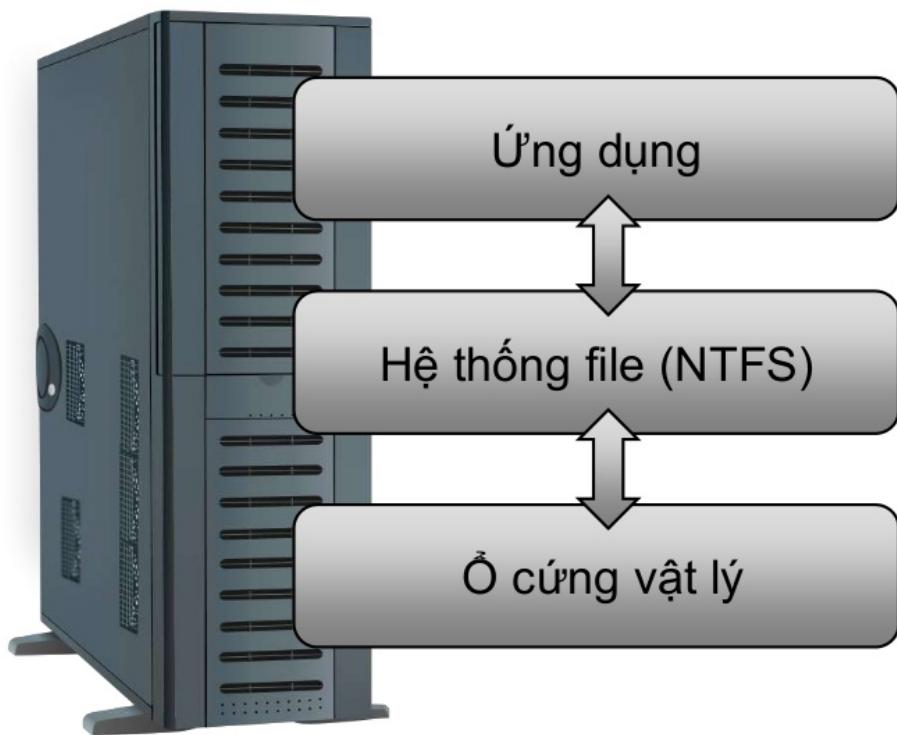
Hadoop Common

- Tập hợp những tiện ích hỗ trợ cho các dự án con của Hadoop
- Bao gồm: tiện ích truy cập hệ thống file, RPC, ...

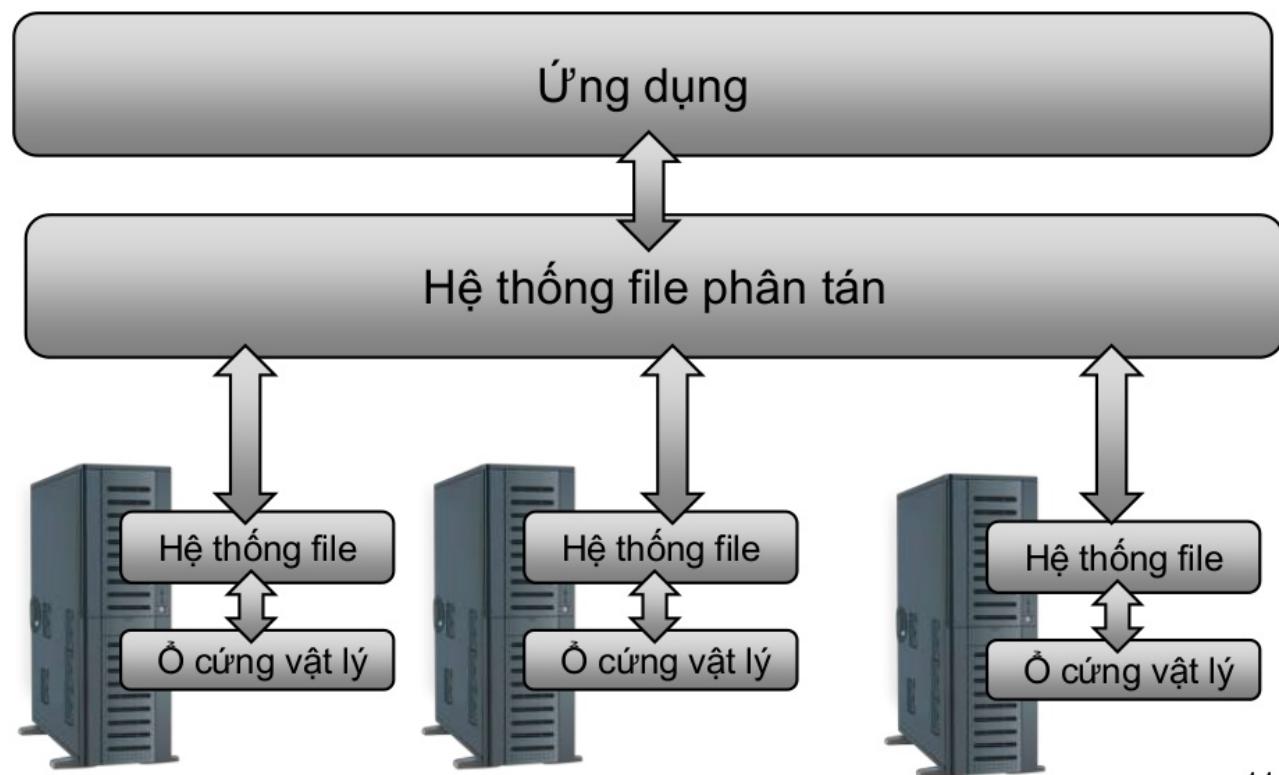
Hadoop Distributed File System

- Hệ thống file phân tán?
- HDFS?
- Kiến trúc của HDFS
- Cách thức lưu trữ và sửa lỗi

Hệ thống file



Hệ thống file phân tán



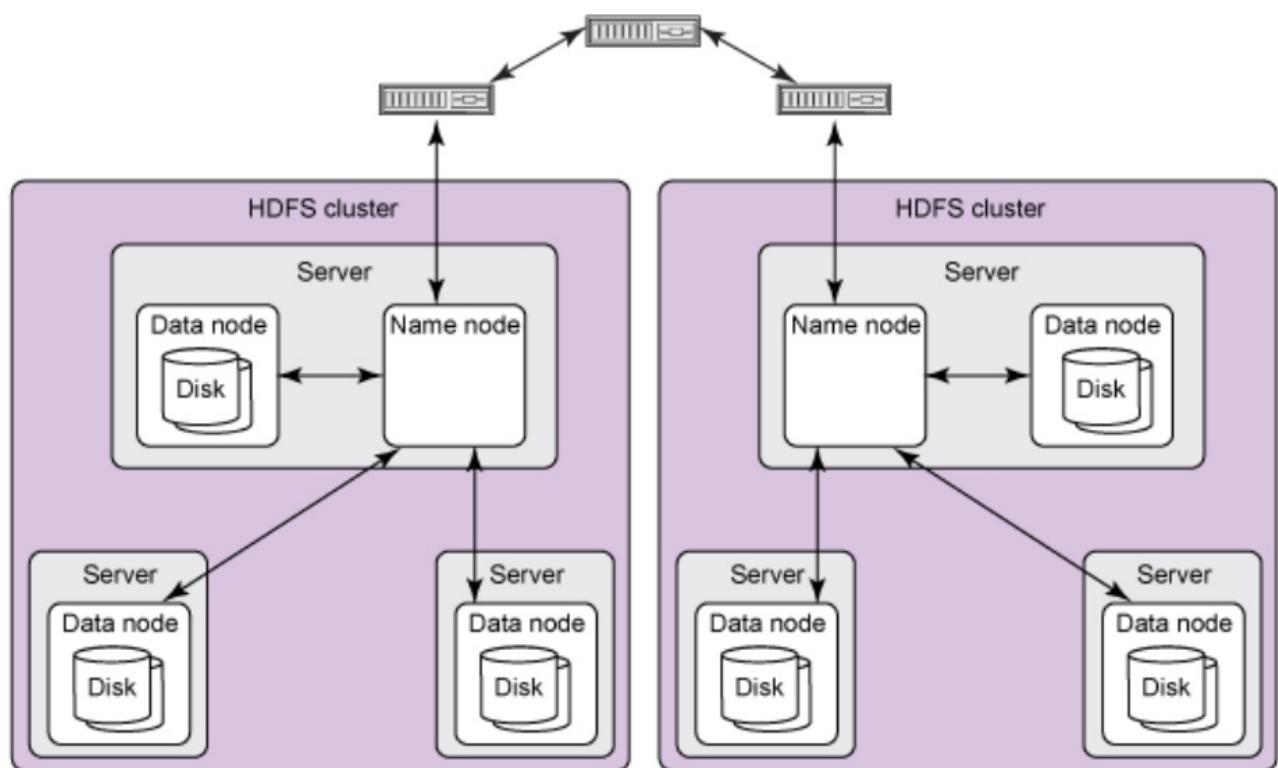
Mục tiêu của HDFS

- Lưu trữ file rất lớn (hàng terabyte)
- Truy cập dữ liệu theo dòng
- Mô hình liên kết dữ liệu đơn giản
 - Ghi 1 lần, đọc nhiều lần
- Di chuyển quá trình xử lý thay vì dữ liệu
- Sử dụng phần cứng phổ thông, đa dạng
- Tự động phát hiện lỗi, phục hồi dữ liệu rất nhanh

Điểm yếu của HDFS

- Ứng dụng cần truy cập với độ trễ cao
 - HDFS tối ưu quá trình truy cập file rất lớn
- Không thể lưu quá nhiều file trên 1 cluster
 - NameNode lưu trên bộ nhớ -> cần nhiều bộ nhớ
- Không hỗ trợ nhiều bộ ghi, sửa dữ liệu bất kỳ

Kiến trúc HDFS

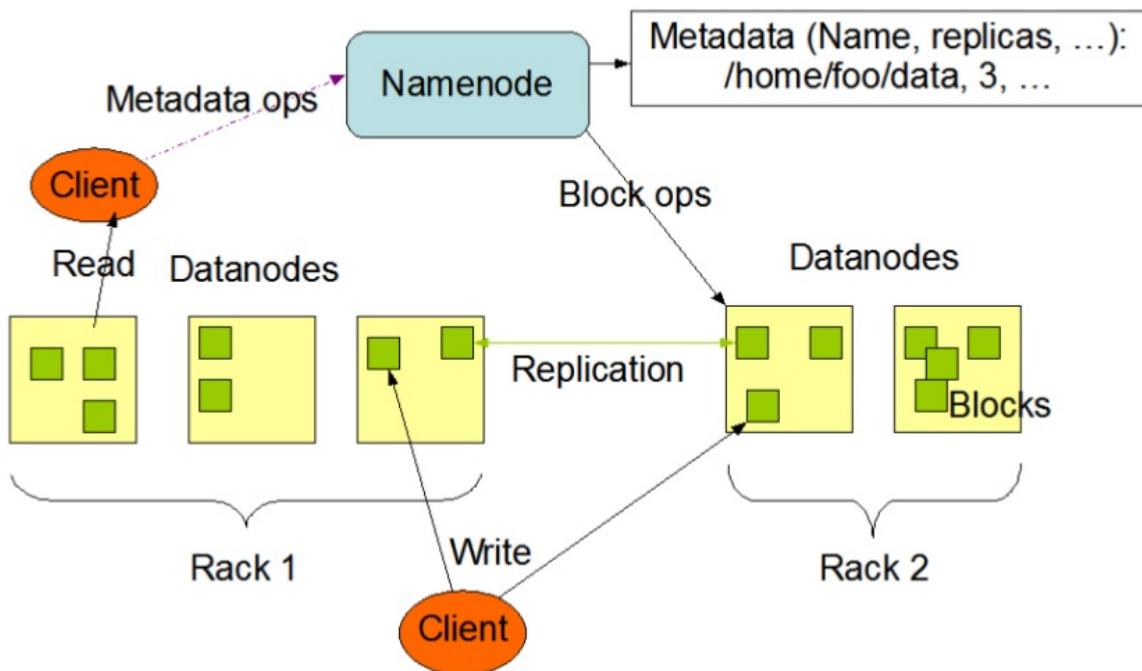


Kiến trúc HDFS (t.t)

- Các khái niệm

- Block: đơn vị lưu trữ dữ liệu nhỏ nhất
 - ✓ Hadoop dùng mặc định 64MB/block
 - ✓ 1 file chia làm nhiều block
 - ✓ Các block chứa ở bất kỳ node nào trong cluster
- NameNode
 - ✓ Quản lý thông tin của tất cả các file trong cluster
- DataNode
 - ✓ Quản lý các block dữ liệu

HDFS Architecture



NameNode

- Thành phần trọng yếu của HDFS
- Quản lý và thực thi các thao tác liên quan đến tên file
 - Đóng, mở, đổi tên
- Quản lý vị trí của các block

DataNode

- Quản lý các block
- Thực hiện thao tác trên dữ liệu
 - Thêm, xóa, nhận biết block
 - Thực hiện các yêu cầu xử lý dữ liệu

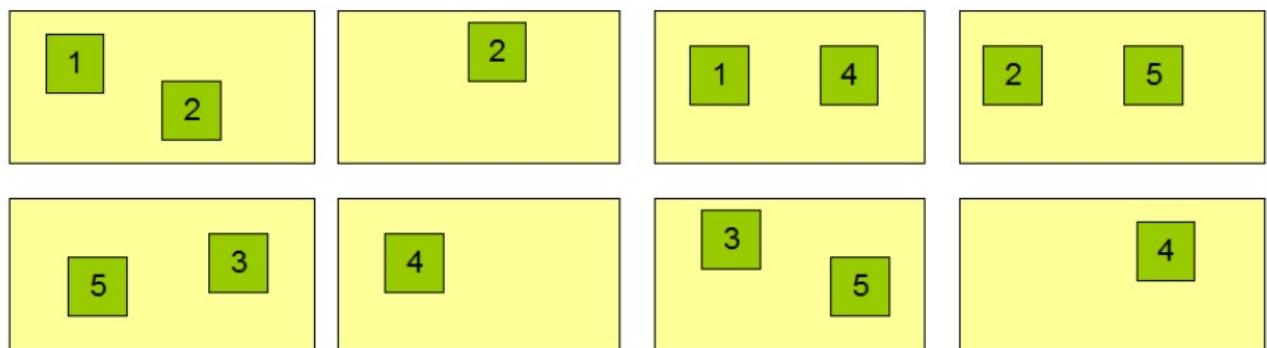
Các thức lưu trữ & phát hiện lỗi

- Bản sao dữ liệu:
 - Mỗi file có nhiều bản sao → nhiều bản sao của block
- NameNode quyết định việc tạo bản sao
 - Nhận dữ liệu Heartbeat & Blockreport từ DataNode
 - ✓ Heartbeat: tình trạng chức năng của DataNode đó
 - ✓ Blockreport: danh sách các block
- Thiết lập chính sách lưu trữ của các bản sao
 - Cơ chế xác định block đó thuộc node nào

Block Replication

Namenode (Filename, numReplicas, block-ids, ...)
/users/sameerp/data/part-0, r:2, {1,3}, ...
/users/sameerp/data/part-1, r:3, {2,4,5}, ...

Datanodes



Chính sách lưu trữ của các bản sao block

- Cực kỳ quan trọng
- Quyết định tính ổn định, an toàn, và khả năng vận hành của hệ thống
- Cần nhiều thời gian và kinh nghiệm
- Quan tâm đến kiến trúc vật lý: rack, bandwidth
- Chính sách thông thường (không tối ưu)
 - Chia block làm 3 bản sao
 - Lưu ở node trong rack nội bộ, 2 block ở 2 node khác nhau trong rack khác (remote rack)

Độ bền vững của HDFS

- Mục tiêu chính: đảm bảo dữ liệu chính xác ngay cả khi lỗi hệ thống xảy ra
- 3 loại lỗi chính:
 - Lỗi ở NameNode
 - Lỗi ở DataNode
 - Sự cản trở của mạng máy tính

Độ bền vững của HDFS (t.t)

- DataNode gửi định kỳ Heartbeat lên NameNode
 - Xác định node bị lỗi nếu NameNode không nhận được Heartbeat.
 - Đưa DataNode khỏi liên kết & cố gắng tạo bản sao khác
- Tái cân bằng cluster
 - Chuyển các block sang DataNode khác có khoảng trống dưới đính mức qui định

Độ bền vững của HDFS (t.t)

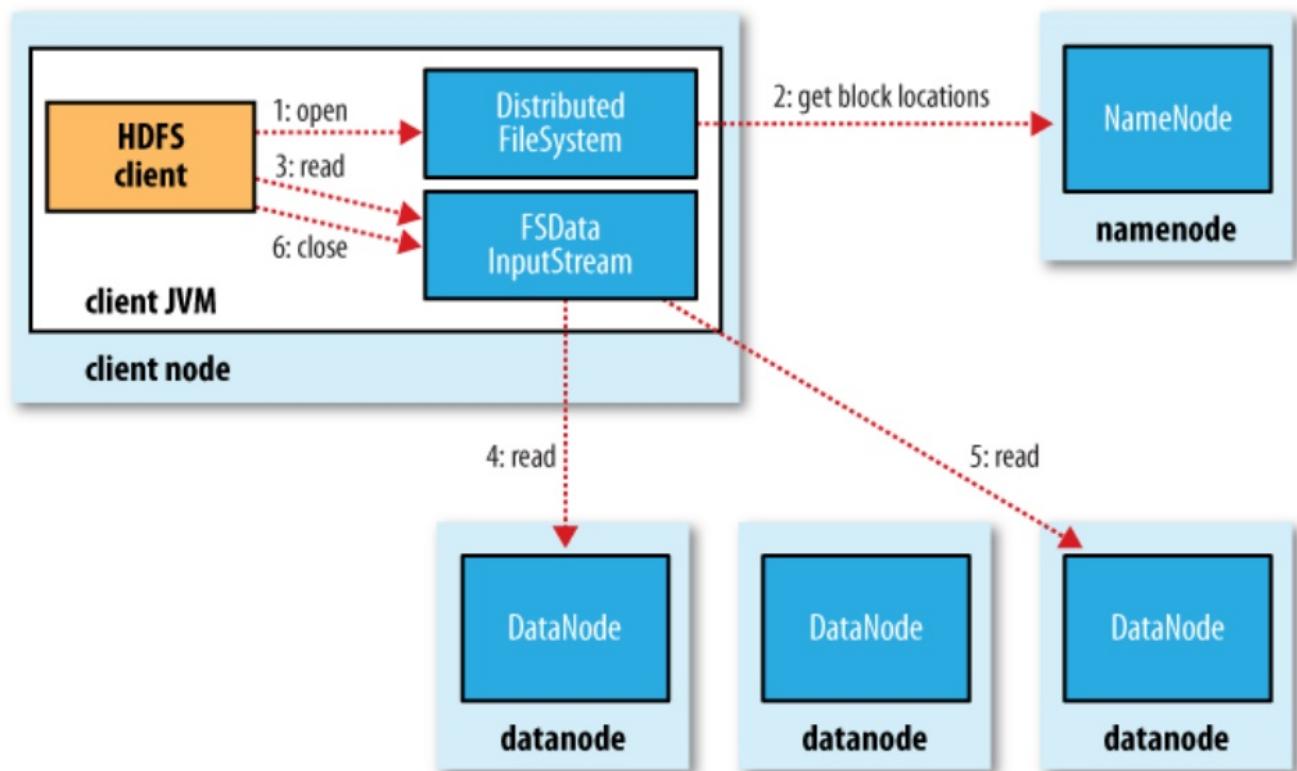
- Lỗi ở NameNode
 - Có thể làm hệ thống HDFS vô dụng
 - Tạo các bản copy của FsImage và EditLog
 - Khi NameNode restart, hệ thống sẽ lấy bản sao gần nhất.

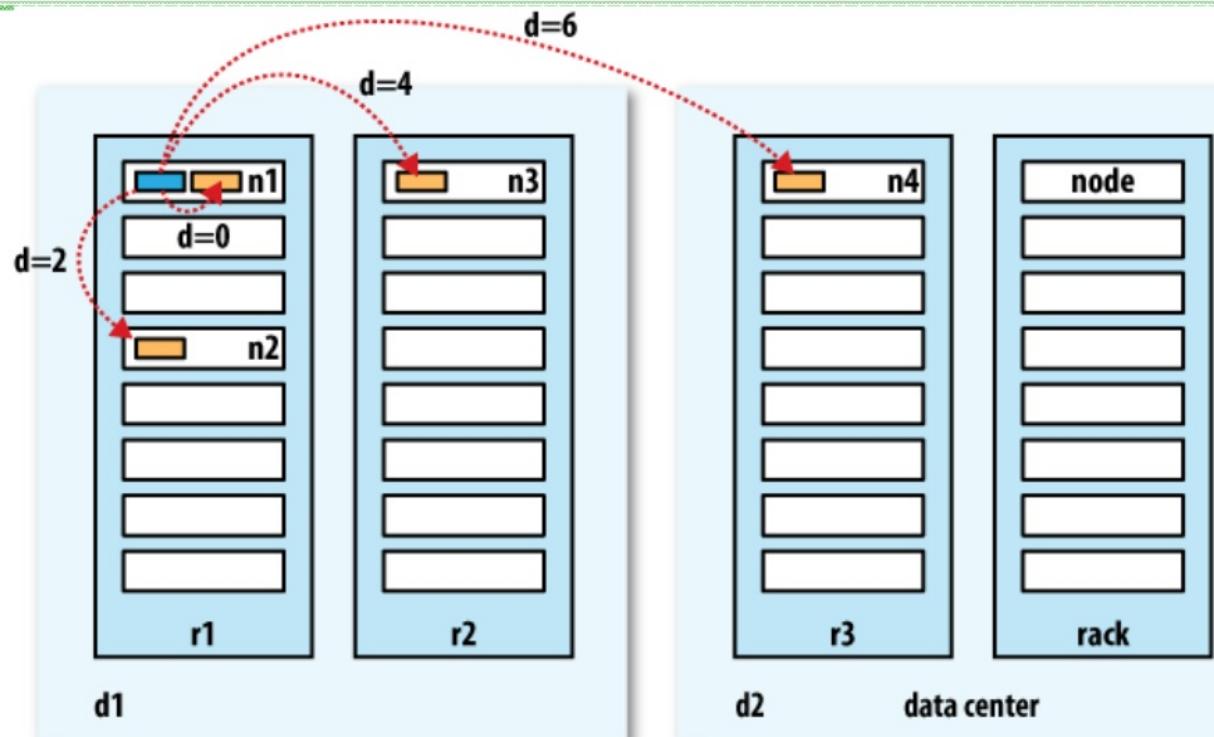
Cơ chế hoạt động

- Đọc dữ liệu:

- Chương trình client yêu cầu đọc dữ liệu từ NameNode
- NameNode trả về vị trí các block của dữ liệu
- Chương trình trực tiếp yêu cầu dữ liệu tại các node.

Cơ chế hoạt động (t.t)



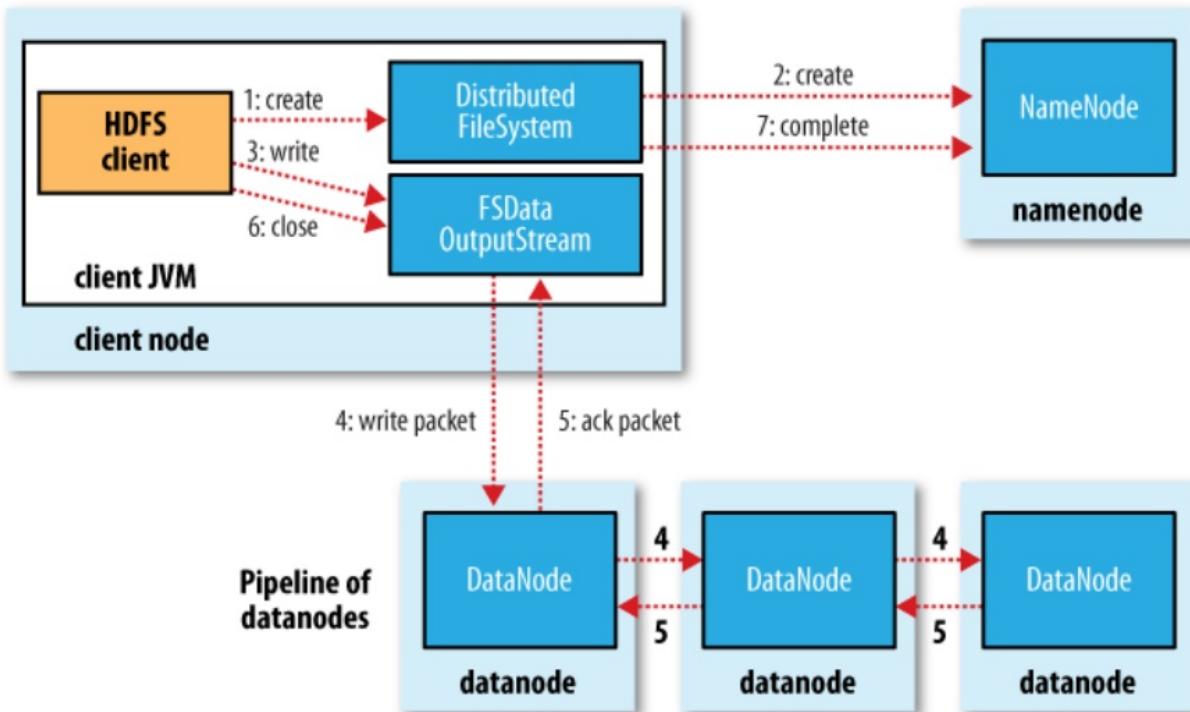


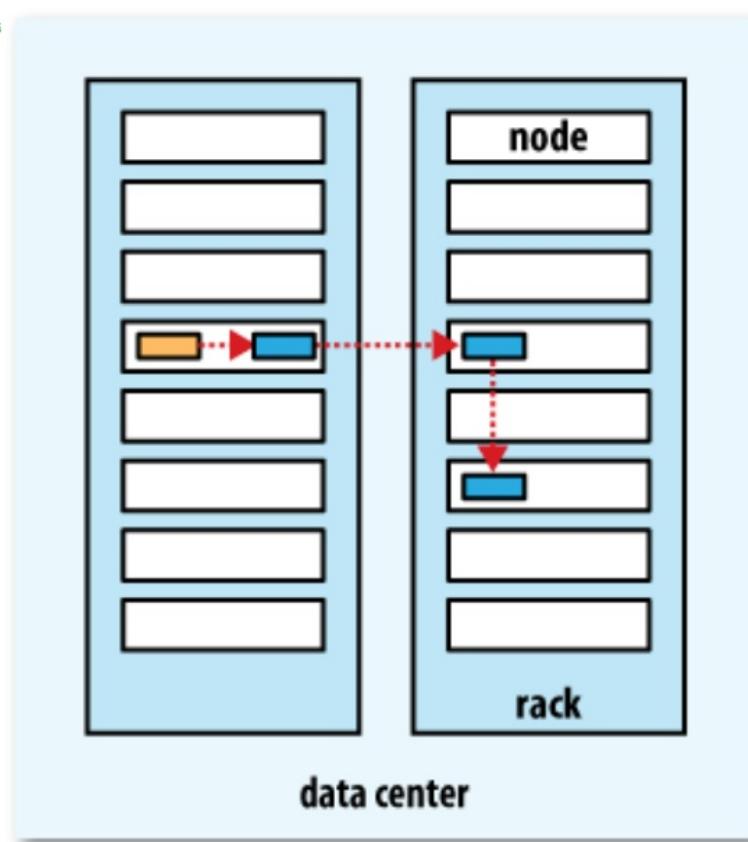
Cơ chế hoạt động (t.t)

- Ghi dữ liệu:

- Ghi theo dạng đường ống (pipeline)
- Chương trình yêu cầu thao tác ghi ở NameNode
- NameNode kiểm tra quyền ghi và đảm bảo file không tồn tại
- Các bản sao của block tạo thành đường ống để dữ liệu tuần tự được ghi vào

Cơ chế hoạt động (t.t)





Map Reduce

- Tại sao cần Map Reduce ?
- Map Reduce là gì ?
- Mô hình Map Reduce
- Thực thi
- Hadoop Map Reduce
- Demo

Tại sao cần Map Reduce ?

- Xử lý dữ liệu với quy mô lớn
 - Muốn sử dụng 1000 CPU
 - ✓ Mong muốn một mô hình quản lý đơn giản
- Kiến trúc Map Reduce
 - Quản lý tiến trình song song và phân tán
 - Quản lý, sắp xếp lịch trình truy xuất I/O
 - Theo dõi trạng thái dữ liệu
 - Quản lý số lượng lớn dữ liệu có quan hệ phụ thuộc nhau
 - Xử lý lỗi
 - Trừu tượng đối với các lập trình viên

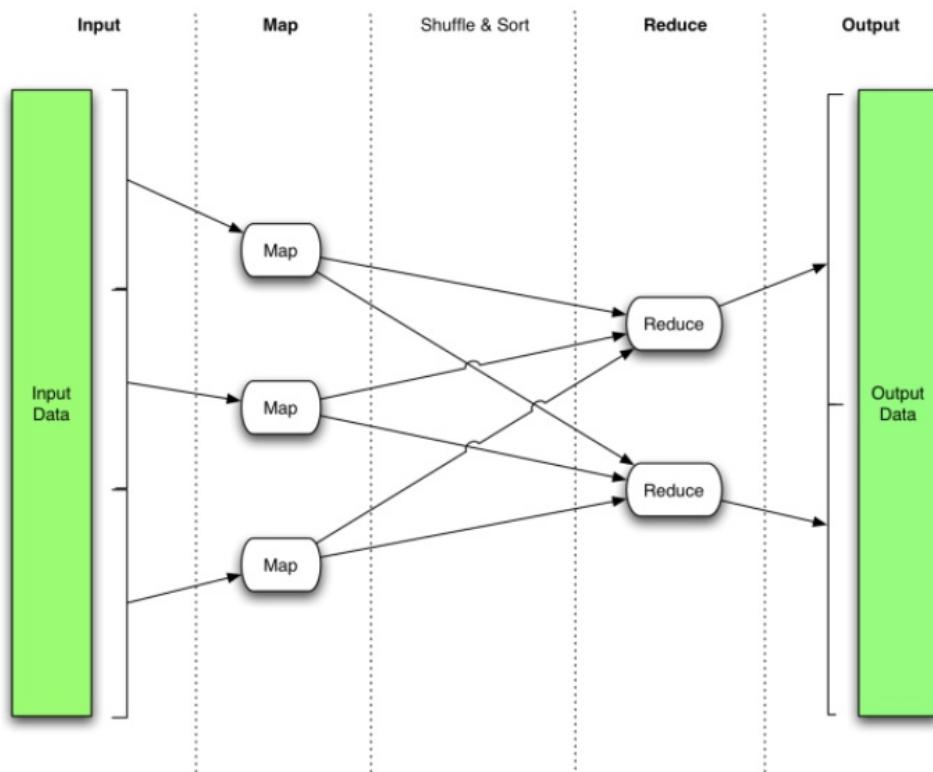
Map Reduce là gì ?

- Mô hình lập trình
 - MapReduce được xây dựng từ mô hình lập trình hàm và lập trình song song
- Hệ thống tính toán phân tán
 - Tăng tốc độ thực thi xử lý dữ liệu
- Giải quyết được nhiều bài toán
- Ân các chi tiết cài đặt, quản lý
 - Quản lý lỗi
 - Gom nhóm và sắp xếp
 - Lập lịch
 -

Map Reduce là gì ?

- Cách tiếp cận : chia để trị
 - Chia nhỏ vấn đề lớn thành các vấn đề nhỏ
 - Xử lý song song từng việc nhỏ
 - Tổng hợp kết quả
- Đọc dữ liệu lớn
- Rút trích thông tin cần thiết từ từng phần tử (Map)
- Trộn và sắp xếp các kết quả trung gian
- Tổng hợp các kết quả trung gian (Reduce)
- Phát sinh kết quả cuối cùng

Map Reduce là gì ?



Mô hình Map Reduce

- Trải qua hai quá trình Map và Reduce
- Map Reduce định nghĩa dữ liệu dưới dạng cặp `<key,value>`
- Map `<k1,v1> -> list(<k2,v2>)`
- Reduce `<k2,list(<v2>)> -> <k3, v3 >`
- (input) `<k1, v1> -> map -> <k2, v2> -> combine -> <k2, v2>`
`-> reduce -> <k3, v3> (output)`

Mô hình Map Reduce

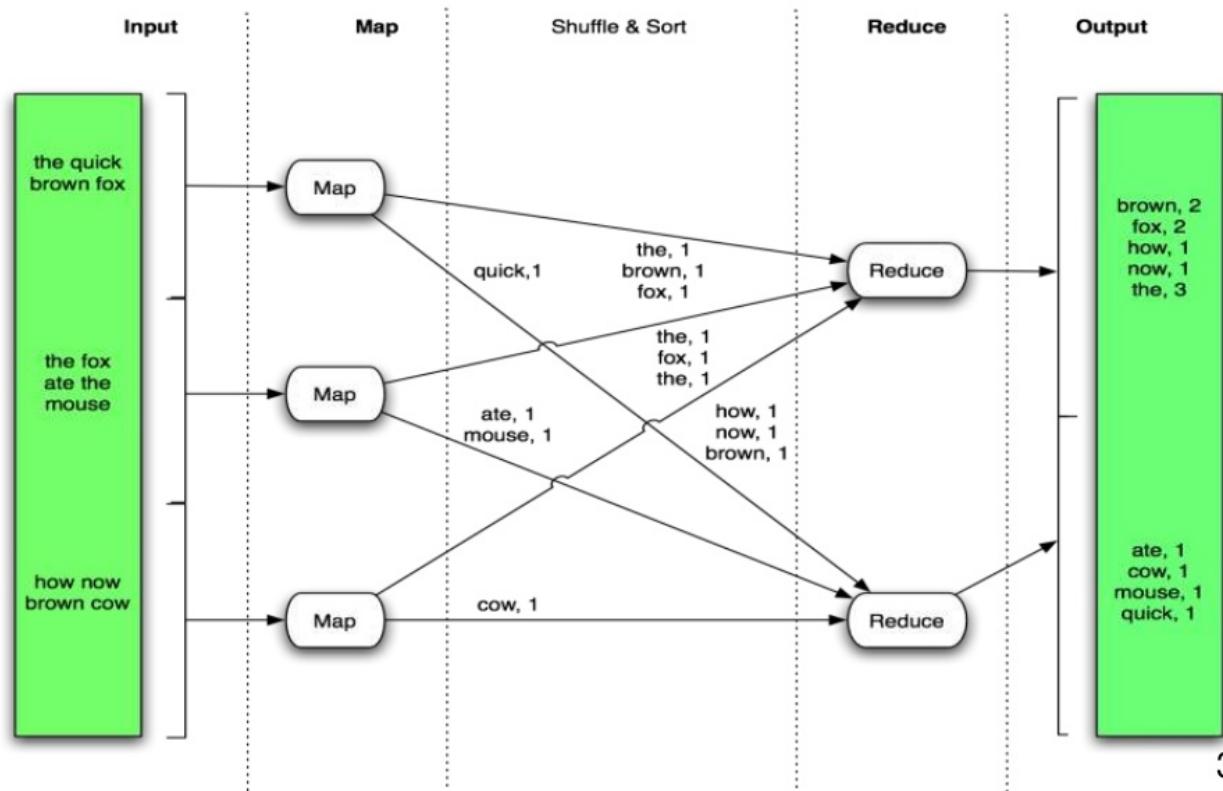
- Hàm Map

- Mỗi phần tử của dữ liệu đầu vào sẽ được truyền cho hàm Map dưới dạng cặp `<key,value>`
- Hàm Map xuất ra một hoặc nhiều cặp `<key,value>`

Mô hình Map Reduce

- Sau quá trình Map, các giá trị trung gian được tập hợp thành các danh sách theo từng khóa
- Hàm Reduce
 - Kết hợp, xử lý, biến đổi các value
 - Đầu ra là một cặp `<key,value>` đã được xử lý

Ví dụ word count



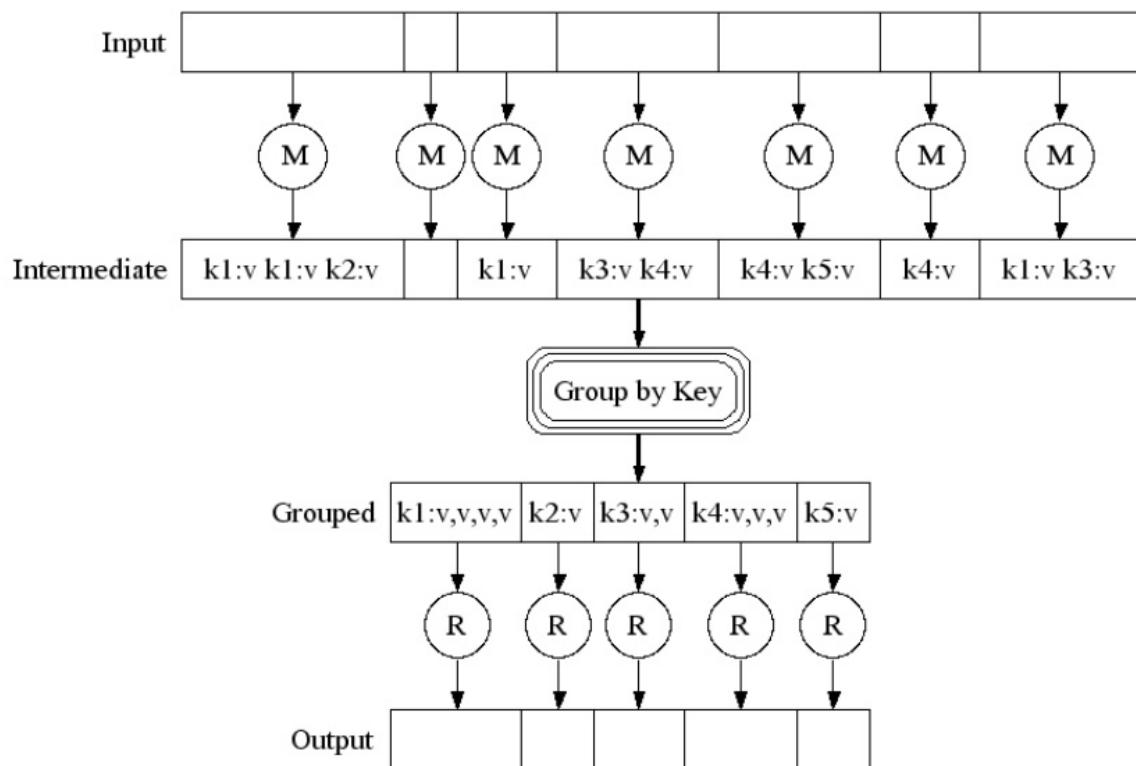
Ví dụ word count (tt)

- Mapper
 - Đầu vào : Một dòng của văn bản
 - Đầu ra : key : từ, value : 1
- Reducer
 - Đầu vào : key : từ, values : tập hợp các giá trị đếm được của mỗi từ
 - Đầu ra : key : từ, value : tổng

Tính toán song song

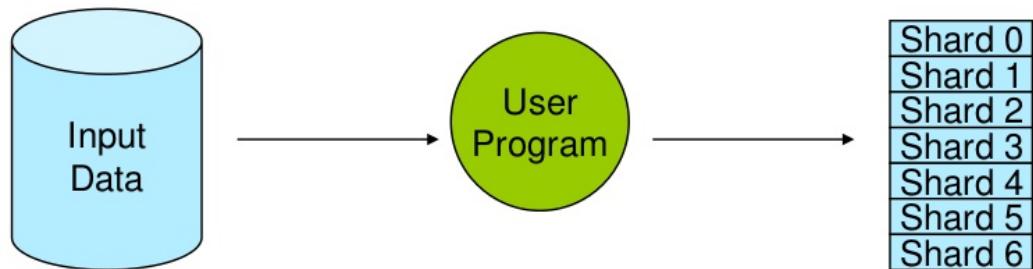
- Hàm Map chạy song song tạo ra các giá trị trung gian khác nhau từ các tập dữ liệu khác nhau
- Hàm Reduce cũng chạy song song, mỗi reducer xử lý một tập khóa khác nhau
- Tất cả các giá trị được xử lý độc lập
- Bottleneck: Giai đoạn Reduce chỉ bắt đầu khi giai đoạn Map kết thúc

Thực thi MR



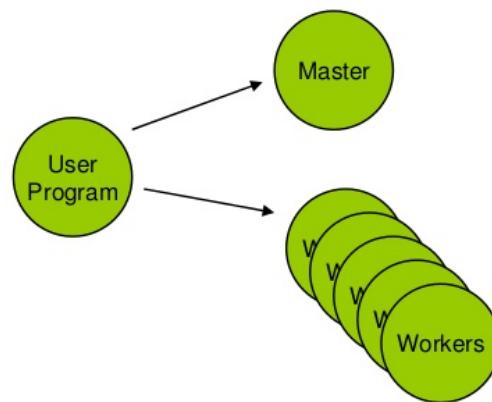
Thực thi (bước 1)

- Chương trình (user program), thông qua thư viện MapReduce phân mảng dữ liệu đầu vào



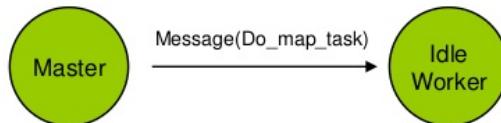
Thực thi (bước 2)

- Map Reduce sao chép chương trình này vào các máy cluster (master và các worker)



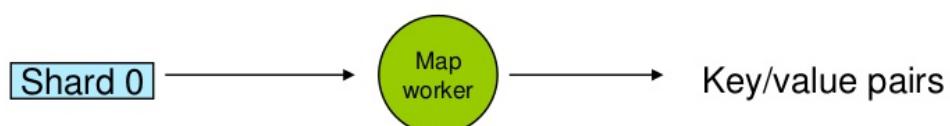
Thực thi (bước 3)

- Master phân phối M tác vụ Map và R tác vụ Reduce vào các worker rảnh rỗi
- Master phân phối các tác vụ dựa trên vị trí của dữ liệu



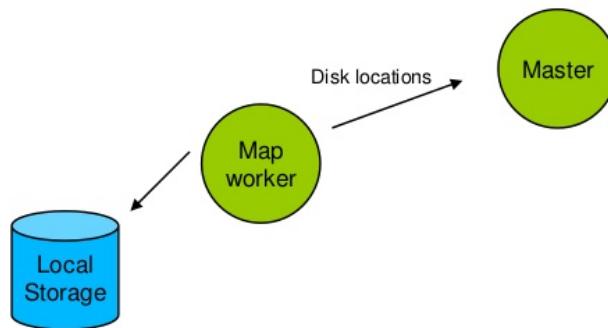
Thực thi (bước 4)

- Mỗi map-task worker đọc dữ liệu từ phân vùng dữ liệu được gán cho nó và xuất ra những cặp `<key,value>` trung gian
 - Dữ liệu này được ghi tạm trên RAM



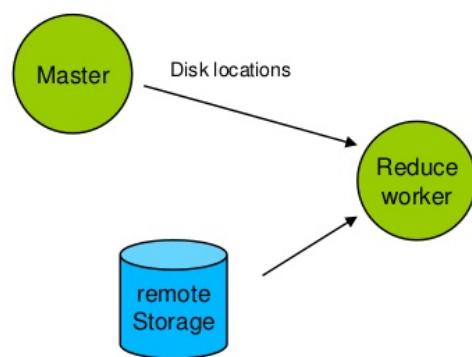
Thực thi (bước 5)

- Mỗi worker phân chia dữ liệu trung gian thành R vùng, lưu xuống đĩa, xóa dữ liệu trên bộ đệm và thông báo cho Master



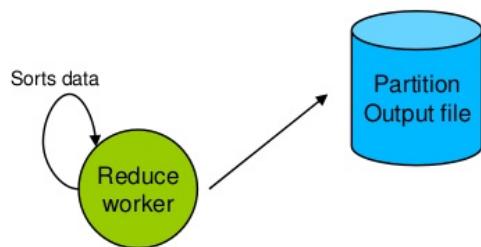
Thực thi (bước 6)

- Master gán các dữ liệu trung gian và chỉ ra vị trí của dữ liệu cho các reduce-task



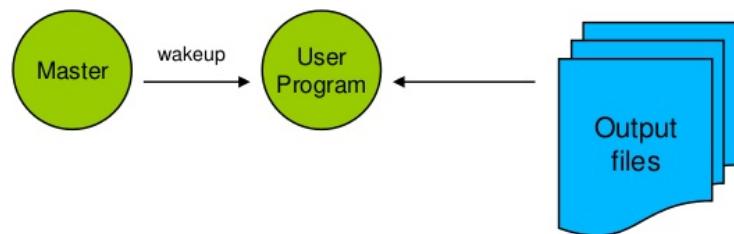
Thực thi (bước 7)

- Mỗi reduce-task worker sắp xếp các key, gọi hàm reduce và xuất kết quả đầu ra



Thực thi (bước 8)

- Master kích hoạt (wakes up) chương trình của người dùng thông báo kết quả hoàn thành
- Dữ liệu đầu ra được lưu trong R tập tin



Hadoop - Map Reduce

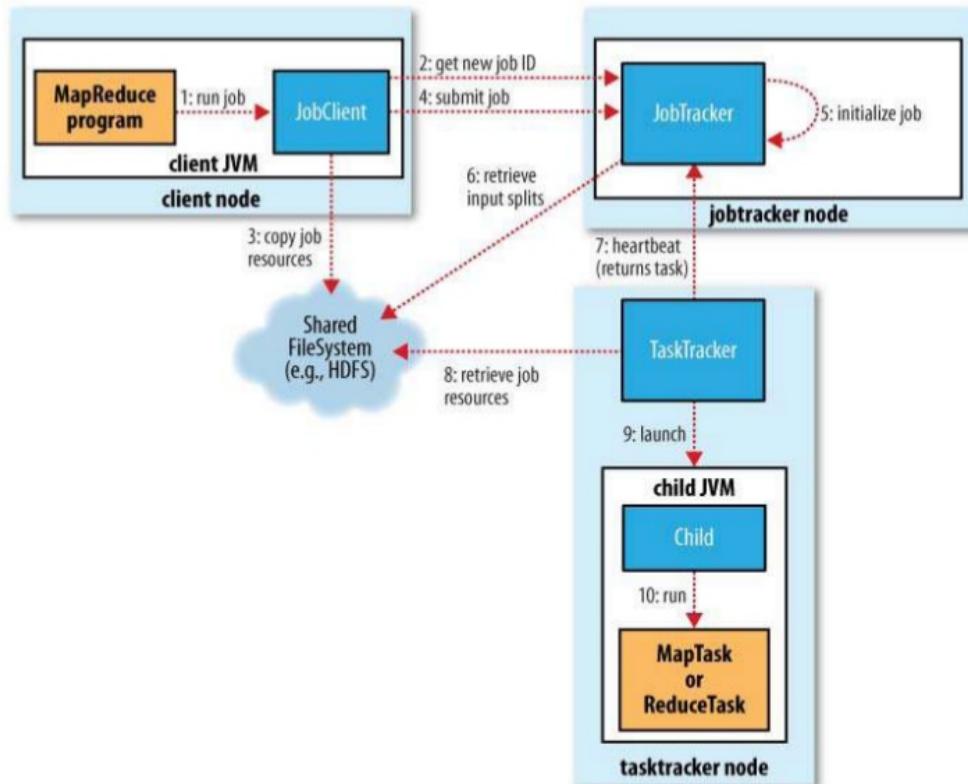
- Là một framework
- Sử dụng HDFS
- Kiến trúc master/slave

	Master	Slave
DFS	Namenode	Datanode
MapReduce	Jobtracker	Tasktracker

Hadoop - Map Reduce

- Client gửi MapReduce Job
- JobTracker điều phối việc thực thi Job
- TaskTracker thực thi các task đã được chia ra

Hadoop - Map Reduce



Job Submission

- Yêu cầu ID cho job mới (1)
- Kiểm tra các thư mục đầu vào và đầu ra
- Chia tách dữ liệu đầu vào
- Chép các tài nguyên bao gồm chương trình (Jar), các tập tin cấu hình, các mảnh dữ liệu đầu vào filesystem của jobtracker (3)
- Thông báo với jobtracker job sẵn sàng để thực thi (4)

Khởi tạo Job

- Thêm job vào hàng đợi & khởi tạo các tài nguyên (5)
- Tạo danh sách các tác vụ (task) (6)

Phân phối các tác vụ

- TaskTracker định kỳ thông báo sẵn sàng nhận các tác vụ mới (7)
- JobTracker giao tác vụ cố định cho TaskTracker (ví dụ 1 TaskTracker chạy đồng thời 2 map-task và 2 reduce-task)

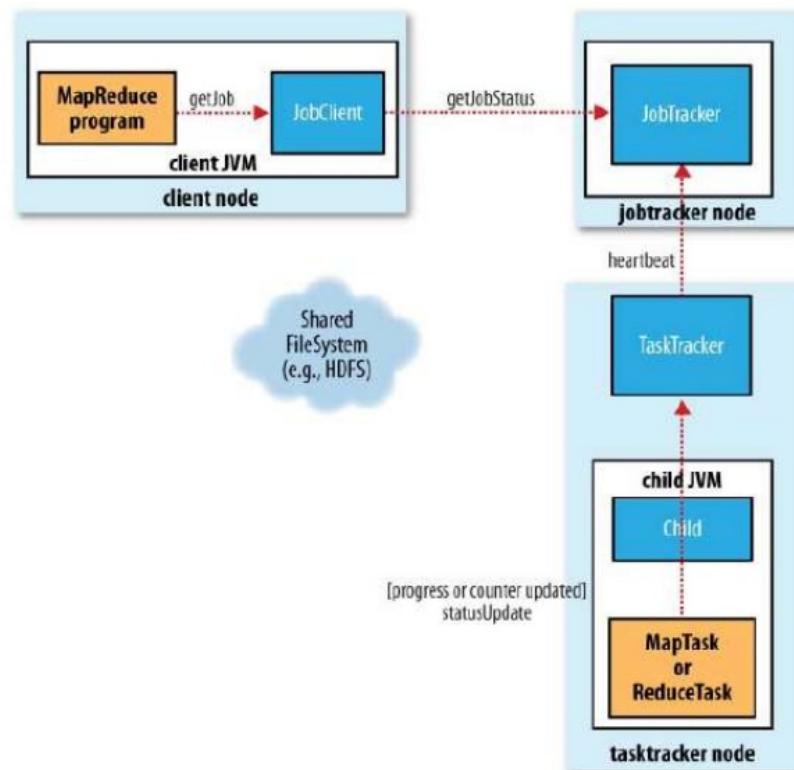
Thực thi tác vụ

- TaskTracker Chép chương trình thực thi (Jar File) và các dữ liệu cần thiết từ hệ thống chia sẻ file
- Tạo tiến trình TaskRunner để thực thi tác vụ

Cập nhật trạng thái

- Cập nhật trạng thái trong quá trình thực thi
 - Tác vụ xử lý được bao nhiêu dữ liệu đầu vào ?
 - Tác vụ hoàn thành thành công ?
 - Tác vụ lỗi ?
- Task process gửi báo cáo 3s một lần cho TaskTracker
- TaskTracker gửi báo cáo 5s một lần cho JobTracker
- JobTracker tổng hợp các báo cáo, gửi lại cho JobClient mỗi giây một lần

Cập nhật trạng thái



Kết thúc Job

- Khi JobTracker nhận được tín hiệu kết thúc của tác vụ cuối cùng
- JobTracker gửi tín hiệu success cho JobClient
- JobClient thông báo cho chương trình của người dùng
- JobTracker thu gom rác, hủy các kết quả trung gian

Khả năng chịu lỗi

- Master phát hiện các lỗi
- Lỗi tác vụ (Task Failure)
 - Văng lỗi ngoại lệ, Bị giết bởi VJM, Treo
 - JobTracker giao cho TaskTracker khác xử lý trong một giới hạn nhất định
 - Hạn chế giao tác vụ mới cho TaskTracker đã xử lý tác vụ bị lỗi

Khả năng chịu lỗi

- Lỗi TaskTracker

- Crashing, Chạy chậm, không gửi báo cáo đúng hạn cho JobTracker
- JobTracker loại bỏ TaskTracker ra khỏi bảng lịch biểu tác vụ (schedule tasks) và thêm vào blacklist
- JobTracker lập lịch lại để chạy các tác vụ đã trao cho TaskTracer bị lỗi

Khả năng chịu lỗi

- Lỗi Jobtracker
 - Nghiêm trọng
 - Chưa có hướng giải quyết

Tối ưu hóa

- Reduce chỉ bắt đầu khi toàn bộ Map kết thúc
 - Đĩa trên một node truy xuất chậm có thể ảnh hưởng tới toàn bộ quá trình
- Băng thông của mạng

Tối ưu hóa

- Đưa ra hàm combiner
 - Có thể chạy trên cùng máy với các mapper
 - Chạy độc lập với các mapper khác
 - Mini Reducer, làm giảm đầu ra của các giai đoạn Map. Tiết kiệm băng thông

Ứng dụng

- Sắp xếp dữ liệu phân tán
- Phân tích thống kê
- Web Ranking
- Dịch máy
- Indexing
- ...

Tổng kết

- Là mô hình đơn giản để xử lý lượng dữ liệu lớn trên mô hình phân tán
- Tập trung vào vấn đề chính cần xử lý

Demo Word Count (1)

- Map

```
public static class MapClass extends MapReduceBase
implements Mapper {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(WritableComparable key, Writable value,
                    OutputCollector output, Reporter reporter)
        throws IOException {
        String line = ((Text)value).toString();
        StringTokenizer itr = new StringTokenizer(line);
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            output.collect(word, one);
        }
    }
}
```

Demo Word Count (2)

- Reduce

```
public static class Reduce extends MapReduceBase
    implements Reducer {
    public void reduce(WritableComparable key, Iterator<IntWritable> values,
                      OutputCollector output, Reporter reporter)
        throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += ((IntWritable) values.next()).get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

Demo Word Count (3)

- Main

```
public static void main(String[] args) throws IOException {  
    //checking goes here  
    JobConf conf = new JobConf();  
  
    conf.setOutputKeyClass(Text.class);  
    conf.setOutputValueClass(IntWritable.class);  
  
conf.setMapperClass(MapClass.class);  
conf.setCombinerClass(Reduce.class);  
conf.setReducerClass(Reduce.class);  
  
    conf.setInputPath(new Path(args[0]));  
    conf.setOutputPath(new Path(args[1]));  
  
    JobClient.runJob(conf);  
}
```