1. **HDFS**
2. **NameNode**

* Trong một Hadoop cluster, chỉ có một Namenode hoạt động tại 1 thời điểm, đóng vai trò như một node master.
* Namenode có nhiệm vụ duy trì và quản lý các DataNode.
* Namenode là nơi lưu trữ và cập nhật MetaData như: logs, tên file, size, vị trí các DataNode. Namenode cũng chỉ dẫn các Datanode thực hiện các thao tác như thêm, xóa, replicate, …
* Namenode luôn lắng nghe và theo dõi để đảm bảo các datanode còn hoạt động, quản lý lượng truy cập tới các datanode và cân bằng dung lượng lưu trữ trong các datanode.

1. **Datanode**

* Các Datanode đóng vai trò trò như các node slave.
* Datanode được sử dụng để lưu trữ data trong hadoop cluster khi dữ liệu được đưa vào hdfs. Một hadoop cluster thường có từ 1 đến 500 hoặc nhiều hơn các Datanodes.
* Datanode là nơi chạy các tiến trình xử lý dữ liệu.
* Các datanode sẽ định kỳ gửi dữ liệu về status của nó tới namenode, mặc định là 3s.

1. **Secondary namenode**

* Secondary namenode đóng vai trò như một node phụ cùng chạy với namenode. Secondary namenode không phải một node dự phòng cho namenode mà có các vai trò và nhiệm vụ rõ ràng.
* Nó thường xuyên đọc các file, các metadata lưu trữ trên RAM của datanode và ghi vào ổ cứng.
* Nó liên tục kiểm tra tính chính xác của các tệp tin lưu trữ trên các datanode.

1. **Block**

* Dữ liệu được đưa vào HDFS sẽ được chia nhỏ thành nhiều block và nằm rải rác ở khắp các nơi trong các datanode. Việc dữ liệu được chia làm bao nhiêu block và mỗi block nằm ở đâu sẽ được quyết định bở metadata trong namenode. Mặc định, mỗi block sẽ có kích thước là 123M(có thể chỉnh sửa).

1. **Replication**

* HDFS có khả năng phục hồi sau lỗi tốt. Data được chia thành nhiều blocks nằm rải rác ở các datanode và các datanode sẽ được replicate thành các bản sao nằm ở các datanode khác nhau. Theo mặc định, mỗi blocks sẽ có 3 bản sao(có thể cấu hình lại).
* Trong trường hợp một datanode gặp sự cố, thì các blocks mà nó đang lưu trữ vẫn có các bản sao nằm ở các datanode khác.
* Việc replicate càng nhiều sẽ càng giảm rủi ro mất mát dữu liệu những cũng chiếm nhiều bộ nhớ hơn.

1. **MapReduce**

MapReduce là một thuật toán hay mô hình được thiết kế để xủa lý các tập dữ liệu lớn song song và phân tán. MapReuce bao gồm 2 pha chính là Map và Reduce.

1. **Map**Hàm Map có nhiệm vụ nhận các Input là các Data block và chia chúng vào các partition và tạo ra các cặp key/value trung gian. Sau đó, nó ghi dữ liệu xuống đĩa cứng và thông báo cho hàm reduce để nhận dữ liệu.
2. **Suffle**

Giữa pha map và reuce là suffle. Sau khi map hoàn thành công việc thì suffle sẽ thu thập và tổng hợp các cặp key/values rồi chuyển qua cho reduce tiếp tục xử lý.

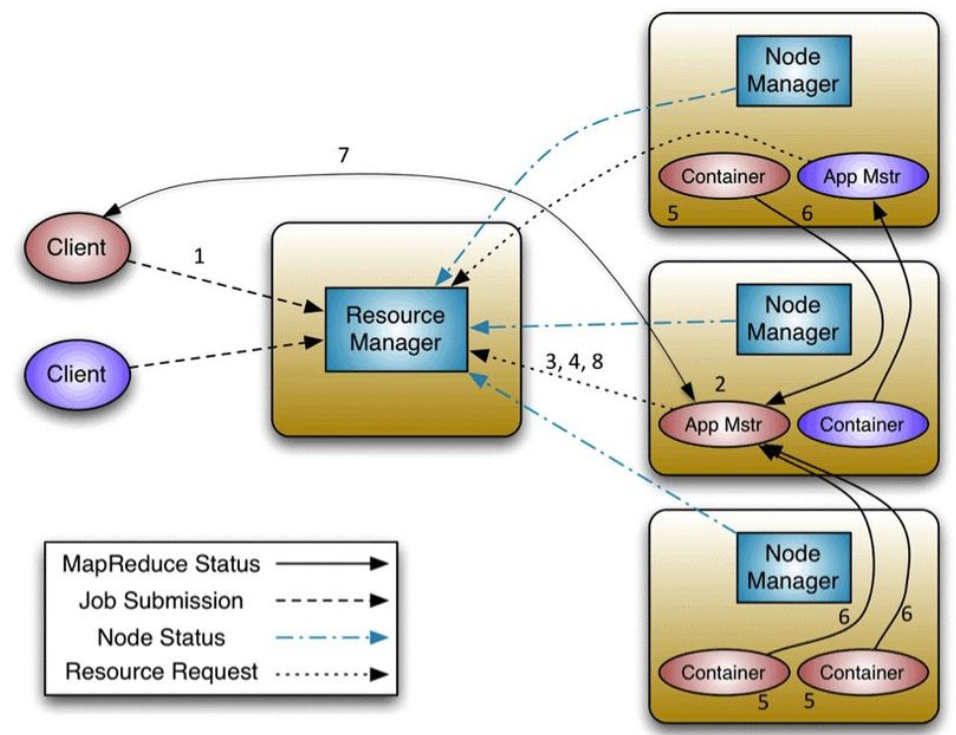
1. **Reduce**

Hàm reduce sẽ tiếp nhận các cặp key/value thực hiện các tác vụ để đưa ra output.

Các bước thực hiện:

1. Tiến hành chuẩn bị các dữ liệu đầu vào để cho Map() có thể xử lý.
2. Lập trình viên thực thi các mã Map() để xử  lý.
3. Tiến hành trộn lẫn các dữ liệu được xuất ra bởi Map() vào trong Reduce Processor.
4. Tiến hành thực thi tiếp mã Reduce() để có thể xử lý tiếp các dữ liệu cần thiết.
5. Thực hiện tạo các dữ liệu xuất ra cuối cùng.
6. **Yarn**

Yarn có 2 chức năng chính là quản lý tài nguyên và theo dõi, lập lịch công việc. Các thành phần chính của yarn bao gồm ResourceManager, NodeManager và ApplicationMaster.



1. **ResourceManager:** có nhiệm vụ quản lý toàn bộ tài nguyên trong cụm, nó cung cấp tài nguyên cho ứng dụng cần dùng, đồng thời quản lý các NodeManager. ResourceManger có 2 thành phần chính là scheduler và ApplicationManager.

* Scheduler: có nhiệm vụ cung cấp tài nguyên cho các ứng dụng.
* ApplicationManager: có nhiệm vụ tiếp nhận các job từ client, cấp phát tài nguyên để start ApplicationMaster cho từng ứng dụng, đồng thời cung cấp dịch vụ restart cho AppicationMaster trong trường hợp bị lỗi.

1. **NodeManger:** được cài trên 1 node có nhiệm vụ quản lý tài nguyên và job trên node đó. NodeManager thường xuyên gửi thông báo về trạng thái đến ResourceManager. NodeManager quản lý các container mà ResourceManager giao cho nó.
2. **ApplicationMaster:** là một container được khởi tạo bởi ResourceManager, tương ứng với mỗi job sumit lên ResourceManager là một ApplicationMaster. ApplicationMaster có nhiệm vụ điều phối các hoạt động của một job trong cụm, xin cấp phát tài nguyên từ ResourceManager và kết nối với NodeManager khác để quản lý các container.
3. **Container:** là tiến trình của yarn, tập hợp các tài nguyên như core, ram. Yarn container được quản lý bở đối tượng Container Lauch Context, đối tượng này lưu trữ các thông tin như biến môi trường, dependencies, security tokens và các chỉ thị để khởi tạo tiến trình. Container cấp quyền cho các ứng dụng chạy trên nó sử dụng một lượng tài nguyên nhất định.

Chi tiết các bước như sau:

1. Client submit job đến ResourceManager bao gồm các bước, tài nguyên cần thiết để khởi tạo ApplicationMaster.
2. ResourceManager cấp phát tài nguyên và start ApplicationMaster.
3. ApplicationMaster đăng ký với ResourceManager cho phép client giao tiếp trực tiếp đến ApplicationMaster.
4. ApplicationMaster sau đó request tài nguyên container từ ResourceManager.
5. Sau khi xin được tài nguyên từ ResourceManager, ApplicationMaster cung cấp cho NodeManager chỉ dẫn để khởi tạo các container trên NodeManager đó. Các container sau đó giao tiếp trực tiếp với ApplicationMaster.
6. ApplicationMaster sẽ liên lạc với các NodeManager ở các node khác để ra yêu cầu công việc cho node này. Node Manager nhận yêu cầu và chạy các task trên container.
7. Việc thực thi job trên container báo cáo lại tiến độ công việc cho ApplicationMaster.
8. Trong khi chương trình thực thi, client giao tiếp trực tiếp với AppMaster để nhận thông tin về tình trạng cung như cập nhật tiến độ công việc.
9. Khi chương trình hoàn tất, ApplicationMaster ngắt kết nối với ResourceManger và off các containers.