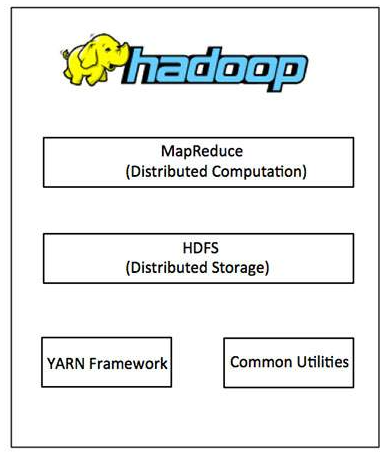
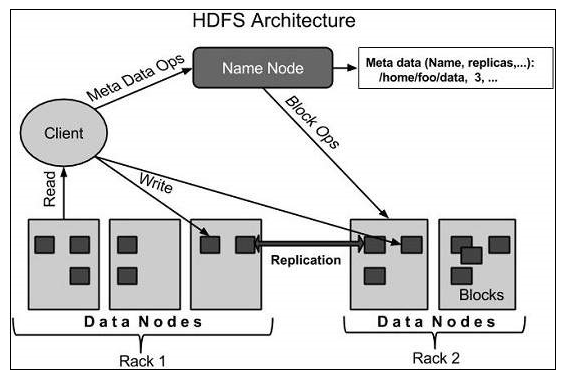
**Bài thu hoạch về Apache Hadoop**

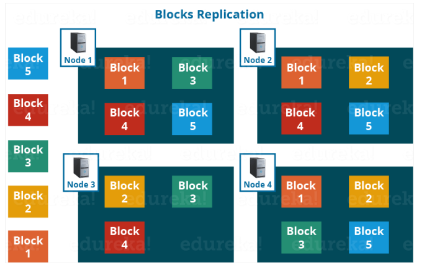
1. **Tổng quan**

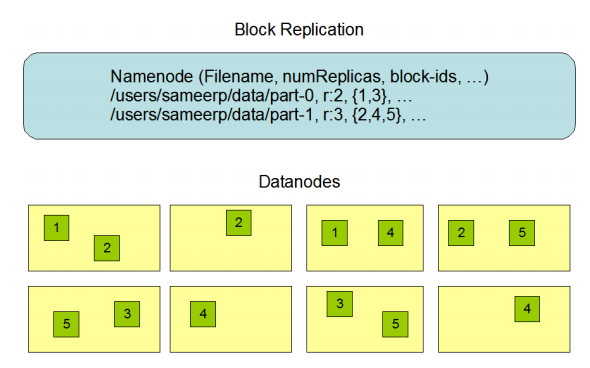
* Apache Hadoop là một mã nguồn mở cho phép sử dụng các distributed processing (ứng dụng phân tán) để quản lý và lưu trữ những tệp dữ liệu lớn. Hadoop áp dụng mô hình MapReduce trong hoạt động xử lý Big Data.



1. **HDFS (Hadoop Distributed File System)**
2. **Kiến trúc HDFS**

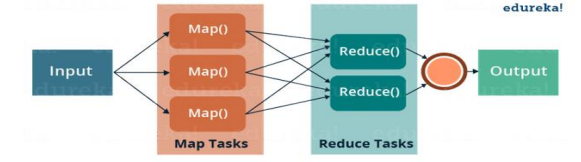
* HDFS là framework cho phép tổ chức file trên hệ thống phân tán của Hadoop
* 
* Cụ thể cơ chế master – slave được thể hiện trong HDFS bằng việc trong một cụm máy, sẽ chỉ bao gồm một máy duy nhất được gọi là là Namenode (Master) và các máy còn lại gọi là Datanode (Slave). Trong đó:
  + Datanode sẽ là nơi lưu trữ các file dữ liệu mà bạn đưa vào.
  + Namenode là nơi lưu địa chỉ của file đó được chia và lưu trên các datanode nào.
* Namenode: chính là master , trong 1 thời điểm chỉ có 1 namenode hoạt động , vì vậy namenode đòi hỏi tính sẵn sàng cực cao, nếu namenode chết là cả cluster sẽ chết – và đây là một điểm yếu chí mạng của Hadoop cluster
  + Namenode là đầu mối truy cập và thực hiện các thao tác với file từ phía client.
  + Namenode có nhiệm vụ duy trì và quản lý các datanode.
  + Namenode là nơi lưu trữ và cập nhật các metadata, ví dụ như địa chỉ của các block trên datanode, quyền truy cập của client,.. Có 2 loại file metadata:
  + FsImage: Là nơi lưu trữ trạng thái của các file system namespace (từ này hơi khó để dịch) kể từ khi namenode được khởi động.
  + EditLogs: Là nơi lưu lại quá sửa đổi của các file system.
  + Nó luôn lắng nghe và theo dõi các datanode để đảm bảo là chúng vẫn “còn sống”.
  + Namenode còn là nơi điều khiển sự cân bằng về dung lượng lưu trữ của các datanode, quản lý lượng truy cập tới các datanode.
* Datanode : chính là các máy slave trong kiến trúc HDFS
  + Là nơi lưu trữ dữ liệu, khi dữ liệu được đưa vào HDFS.
  + Là nơi chạy các tiến trình xử lý dữ liệu (các job).
  + Gửi dữ liệu định kỳ về tình trạng “sức khỏe” tới datanode, mặc định là cứ 3s nó sẽ gửi một lần.
* Secondary namenode : node phụ chạy cùng máy namenode
  + 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.
* Block :
  + Dữ liệu được đưa vào trong HDFS sẽ được chia nhỏ thành nhiều mảnh và nằm rải rác khắp nơi trong các datanode, mỗi mảnh dữ liệu như vậy được gọi là một block. Việc dữ liệu được chia làm bao nhiêu block và mỗi block nằm ở datanode nào sẽ được lưu ở metadata namenode.
  + Trong HDFS, mỗi block có dung lượng tối đa là 128M (có thể cấu hình được để điều chỉnh lại). Nghĩa là nếu bạn có một file example.txt nặng 514M, thì khi đưa vào trong HDFS, nó sẽ được chia làm 5 block, với 4 block đầu tiên mỗi block nặng 128M và block cuối cùng nặng 2M
* 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
* Replication Management : Dữ liệu sau khi được chia thành các block, ngoài việc được lưu rải rác trên khắp các datanode, các block còn được nhân bản thành nhiều phiên bản khác nhau và lưu tại các datanode khác nhau. Bạn có thể xem hình ảnh dưới đây để hiểu rõ hơn



* + - Block 1 được lưu tại node 1, node 2, node 4.
    - Block 2 được lưu tại node 2, node 3, node 4
    - Block 3 được lưu tại node 1, node 3, node 4
    - Block 4 được lưu tại node 1, node 2, node 3
    - Block 5 được lưu tại node 1, node 2, node 4
  + Giả sử node 1 gặp sự cố, thì các dữ liệu mà node đang lưu (block 1, 3, 4, 5) vẫn có lấy được từ các node khác. Điều này tương tự các node khácBằng cơ chế “Quản lý các bản sao” mà HDFS có cách sắp xếp và nhân bản các block để hạn chế tối đa việc mất mát dữ liệu khi có datanode gặp sự cố
* Nhân bản dữ liệu :
  + HDFS được thiết kế để lưu trữ các file cực lớn một cách tin cậy trên nhiều máy trong một cụm máy chủ lớn. Nó chia file thành một chuỗi các block, tất cả các block có kích thước bằng nhau trừ block cuối cùng. Các block này được nhân bản để chống lỗi theo hệ thóng nhân bản được cấu hình. Các file trong HDFS được ghi một lần duy nhất và không thể thay đổi.
  + NameNode thực hiện mọi thao tác đều dựa vào các nhân bản của block. Nó định kỳ nhận heartbeat từ các DataNode để xác định tính sẵn sàng của DataNode và báo cáo BlockReport về các block được lưu trên DataNode đó. 
* Thu hồi vùng nhớ
  + Xóa file và khôi phục file đã xóa
    - khi một file bị xóa, nó sẽ không bị xóa ngay mà thay vào đó nó sẽ bị đổi tên và di chuyển vào thư mục trash. File có thể được khôi phục bất cứ lúc nào khi file vẫn còn tồn tại trong trash. Sau khi hết thời gian ở trong trash, HSFS sẽ tự động xóa file đó khỏi hệ thống. Việc xóa file sẽ giải phóng vùng nhớ liên kết tới file. Sẽ có một khoảng thời gian trễ từ lúc người dùng xóa file cho đến khi bộ nhớ trống của HDFS được tăng lên.
  + Giảm hệ số nhân bản:
    - Khi hệ số nhân bản giảm, NameNode sẽ xác định những nhân bản thừa để thực hiện xóa bỏ. Thông tin này nhận được từ các bản tin của DataNode. DataNode sẽ thực hiện xóa bỏ block và giải phóng vùng nhớ tương ứng. Tuy nhiên vẫn có độ trễ giữa thời gian xóa và thời gian vùng nhớ trống được tăng lên.

1. **MapReduce**

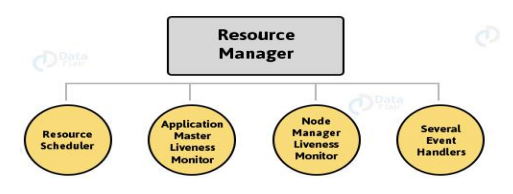
* Mapreduce là lớp xử lý của Hadoop . Mô hình lập trình MapReduce được thiết kế để xử lý song song khối lượng lớn dữ liệu bằng cách chia công việc thành một tập hợp các nhiệm vụ độc lập.
* Luồng dữ liệu và cách hoạt động Mapreduce



* Input files: Dữ liệu cho một tác vụ MapReduce được lưu trữ trong các tệp đầu vào và các tệp đầu vào thường nằm trong HDFS . Định dạng của các tệp này là tùy ý, trong khi các tệp nhật ký dựa trên dòng và định dạng nhị phân cũng có thể được sử dụng.
* InputFormat: Xác định cách các tệp đầu vào này được phân chia và đọc. Nó chọn các tệp hoặc các đối tượng khác được sử dụng để nhập liệu. InputFormat tạo InputSplit.
* InputSplits: Nó được tạo bởi InputFormat, đại diện một cách logic cho dữ liệu sẽ được xử lý bởi một riêng lẻ Mapper. Một nhiệm vụ Mapper được tạo cho mỗi lần tách; do đó số lượng nhiệm vụ
* Mapper sẽ bằng số lượng InputSplits.
* RecordReader : Chuyển đổi dữ liệu thành các cặp key-value phù hợp để đọc. Theo mặc định, nó sử dụng TextInputFormat để chuyển đổi dữ liệu thành một cặp khóa-giá trị. RecordReader giao tiếp với InputSplit cho đến khi việc đọc tệp không được hoàn thành.
* Mapper: Nó xử lý từng bản ghi đầu vào (từ RecordReader) và tạo ra cặp key-value mới và cặp key-value được tạo bởi Mapper này hoàn toàn khác với cặp đầu vào. Đầu ra của Mapper còn được gọi là đầu ra trung gian được ghi vào đĩa cục bộ. Đầu ra của Mapper không được lưu trữ trên HDFS vì đây là dữ liệu tạm thời và việc ghi trên HDFS sẽ tạo ra các bản sao không cần thiết.
* Combiner: Hadoop MapReduce Combiner thực hiện tổng hợp cục bộ trên đầu ra của người lập map, giúp giảm thiểu việc truyền dữ liệu giữa trình ánh xạ. Khi chức năng bộ kết hợp được thực thi, kết quả đầu ra sau đó sẽ được chuyển cho trình phân vùng để làm việc tiếp theo.
* Partitioner: Trình phân vùng lấy đầu ra từ các bộ kết hợp và thực hiện phân vùng. Phân vùng đầu ra diễn ra trên cơ sở khóa và sau đó được sắp xếp.
* Shuffling and Sorting : Khi tất cả các trình ánh xạ đã hoàn thành và đầu ra của chúng được xáo trộn trên các nút giảm tốc, thì đầu ra trung gian này được hợp nhất và sắp xếp, sau đó được cung cấp làm đầu vào để giảm giai đoạn.
* Reduce: Nó lấy tập hợp các cặp key-value trung gian do người lập map tạo ra làm đầu vào và sau đó chạy một hàm giảm thiểu trên mỗi cặp để tạo ra đầu ra. Đầu ra của bộ giảm tốc là đầu ra cuối cùng, được lưu trữ trong HDFS.
* RecordWriter: Nó ghi cặp key-value đầu ra này từ pha Giảm tốc vào các tệp đầu ra
* Output Format: Cách các cặp khóa-giá trị đầu ra này được ghi trong tệp đầu ra bởi RecordWriter được xác định bởi OutputFormat. Đầu ra cuối cùng của bộ giảm tốc được ghi trên HDFS bởi OutputFormat

1. **YARN** **(Yet Another Resource Locator)**

* Kiến trúc YARN về cơ bản tách lớp quản lý tài nguyên khỏi lớp xử lý. Trong phiên bản Hadoop 1.0, trách nhiệm của Trình theo dõi công việc được phân chia giữa người quản lý tài nguyên và người quản lý ứng dụng.
* YARN cũng cho phép các công cụ xử lý dữ liệu khác nhau như xử lý đồ thị, xử lý tương tác, xử lý luồng cũng như xử lý hàng loạt để chạy và xử lý dữ liệu được lưu trữ trong HDFS (Hệ thống tệp phân tán Hadoop), do đó làm cho hệ thống hiệu quả hơn nhiều. Thông qua các thành phần khác nhau, nó có thể phân bổ động các tài nguyên khác nhau và lên lịch xử lý ứng dụng. Để xử lý dữ liệu khối lượng lớn, việc quản lý các tài nguyên sẵn có một cách hợp lý để mọi ứng dụng có thể tận dụng chúng là điều khá cần thiết.
* **Tính năng của YARN**: YARN trở nên phổ biến vì các tính năng sau:
  + Khả năng mở rộng: Bộ lập lịch trong Trình quản lý tài nguyên của kiến ​​trúc YARN cho phép Hadoop mở rộng và quản lý hàng nghìn nodes và clusters.
  + Khả năng tương thích: YARN hỗ trợ các ứng dụng thu nhỏ bản đồ hiện có mà không bị gián đoạn, do đó, nó cũng tương thích với Hadoop 1.0.
  + Multi-tenancy: Nó cho phép nhiều công cụ truy cập
* **Các thành phần chính của kiến ​​trúc YARN bao gồm :**
  + Client: Gửi các map-reduce job
  + Resource Manager: Nó là daemon chính của YARN và chịu trách nhiệm phân công và quản lý tài nguyên giữa tất cả các ứng dụng. Bất cứ khi nào nó nhận được một yêu cầu xử lý, nó sẽ chuyển tiếp nó đến người quản lý nút tương ứng và phân bổ tài nguyên để hoàn thành yêu cầu đó. Nó có hai thành phần chính:



* + - Scheduler: có trách nhiệm phân bổ tài nguyên cho các ứng dụng khác nhau. Đây là Scheduler thuần túy vì nó không thực hiện theo dõi trạng thái cho ứng dụng. Nó cũng không sắp xếp lại các tác vụ bị lỗi do lỗi phần cứng hoặc phần mềm. Bộ lập lịch phân bổ các tài nguyên dựa trên các yêu cầu của ứng dụng
    - ApplicationManager : Chấp nhận nộp công việc. Đàm phán container đầu tiên để thực thi ApplicationMaster. Một nơi chứa kết hợp các yếu tố như CPU, bộ nhớ, đĩa và mạng Khởi động lại container ApplicationMaster khi không thành công.
  + Node manager: Nó quản lý từng node riêng lẻ trên cụm Hadoop và quản lý ứng dụng và quy trình làm việc cũng như nút cụ thể đó. Công việc chính của nó là theo kịp với Người quản lý tài nguyên. Nó đăng ký với Resource Manager và gửi nhịp tim với tình trạng sức khỏe của nút. Nó giám sát việc sử dụng tài nguyên, thực hiện quản lý nhật ký và cũng giết một vùng chứa dựa trên chỉ dẫn từ trình quản lý tài nguyên. Nó cũng chịu trách nhiệm tạo tiến trình vùng chứa và bắt đầu nó theo yêu cầu của Application master.
  + Application Master: Application master chịu trách nhiệm thương lượng tài nguyên với người quản lý tài nguyên, theo dõi trạng thái và giám sát tiến độ của một ứng dụng duy nhất. Application master yêu cầu vùng chứa từ trình quản lý nút bằng cách gửi Bối cảnh khởi chạy vùng chứa (CLC) bao gồm mọi thứ mà ứng dụng cần để chạy. Khi ứng dụng được khởi động, nó sẽ gửi báo cáo tình trạng đến người quản lý tài nguyên theo thời gian.
  + Container: Nó là một tập hợp các tài nguyên vật lý như RAM, lõi CPU và đĩa trên một nút duy nhất. Các vùng chứa được gọi bởi Ngữ cảnh khởi chạy vùng chứa (CLC) là một bản ghi có chứa thông tin như biến môi trường, mã thông báo bảo mật, phụ thuộc, v.v.