**BÁO CÁO TUẦN 3**

[1. Hadoop 1](#_Toc14507034)

[2. HDFS 3](#_Toc14507035)

[3. RDD (Resilient Distributed DataSet) 5](#_Toc14507036)

[4. Apache Spark 7](#_Toc14507037)

# Hadoop

* Hadoop là một Apache framework mã nguồn mở được viết bằng java, cho phép xử lý phân tán các tập dữ liệu lớn trên các cụm máy tính (clusters of computers) thông qua mô hình lập trình đơn giản. Hadoop được thiết kế để mở rộng quy mô từ một máy chủ đơn sang hàng ngàn máy tính khác có tính toán và lưu trữ cục bộ (local computation and storage).
* Hadoop framework gồm 2 phần chính:
* Hadoop Distributed File System (HDFS): Đây là hệ thống file phân tán cung cấp truy cập thông lượng cao cho ứng dụng khai thác dữ liệu.
* Hadoop MapReduce:[[1]](#footnote-2)[[2]](#footnote-3)

+ Đây là hệ thống dựa trên YARN dùng để xử lý song song các tập dữ liệu lớn (chia vấn đề thành các task nhỏ hơn, ánh xạ các task này sang các bộ xử lý song song trên mạng rồi reduce các câu trả lời trung gian)

+ Gồm 2 pha:

Hàm Map: Đọc dữ liệu từ HDFS, xử lý một cặp (key, value) để sinh ra một cặp (keyI, valueI) - key và value trung gian. Dữ liệu này input vào hàm Reduce.   
Hàm Reduce: Tiếp nhận các (keyI, valueI) và trộn các cặp (keyI, valueI) trung gian, lấy ra các valueI có cùng keyI.

+ Ý tưởng

Chia vấn đề cần xử lý thành các phần nhỏ để xử lý.

Xử lý các phần nhỏ đó một cách song song và độc lập trên các máy tính phân tán.

Tổng hợp các kết quả thu được để dưa ra kết quả cuối cùng.

+ Không hiệu quả với các trường hợp tính toán nhiều lượt và sử dụng nhiều thuật toán.

* Cách thức hoạt động:
* Giai đoạn 1: Một user hay một ứng dụng có thể submit một job lên Hadoop (hadoop job client) với yêu cầu xử lý cùng các thông tin cơ bản:

+ Nơi lưu (location) dữ liệu input, output trên hệ thống dữ liệu phân tán.

+ Các java class ở định dạng jar chứa các dòng lệnh thực thi các hàm map và reduce.

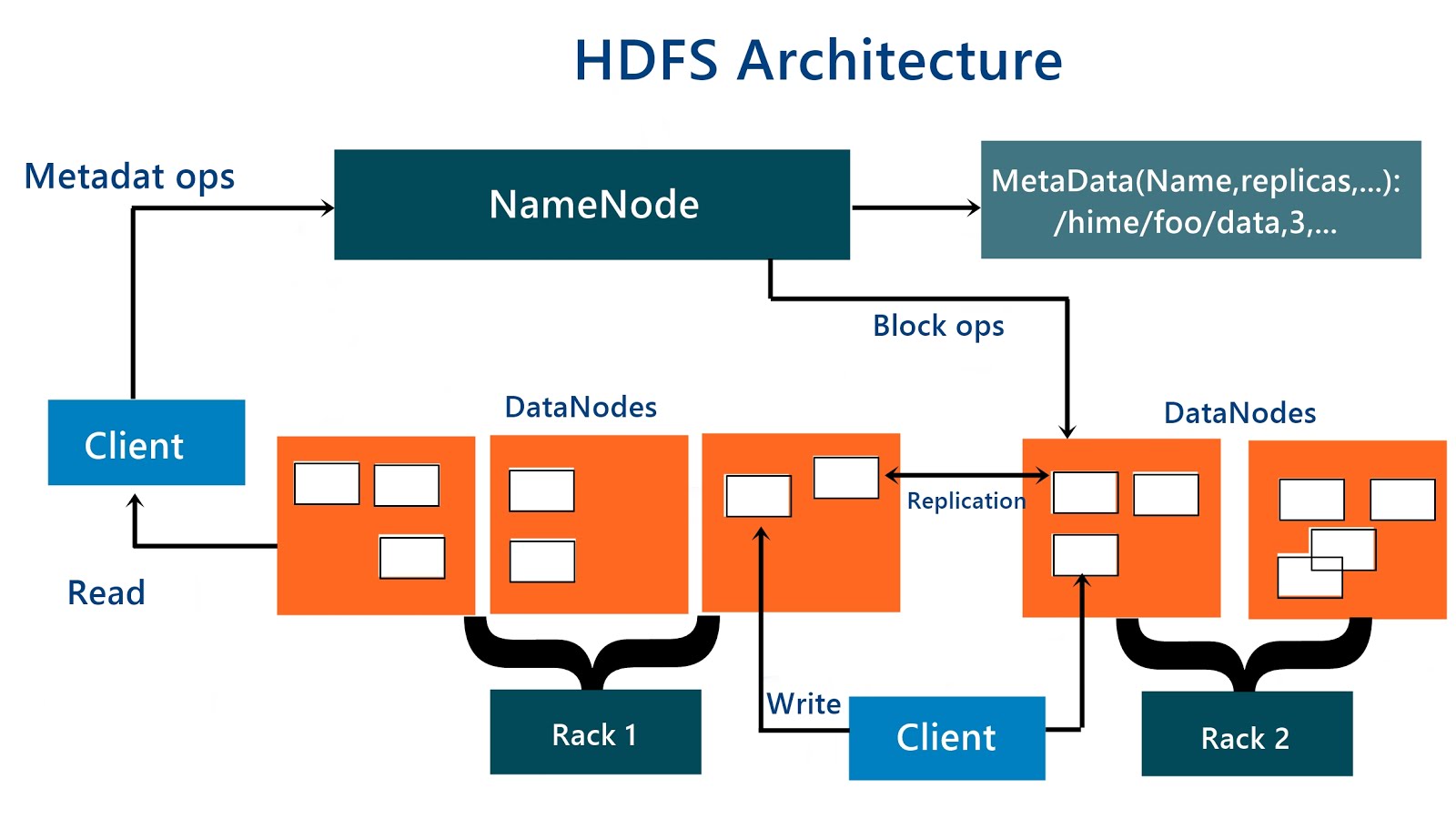
+ Các thiết lập cụ thể liên quan đến job thông qua các thông số truyền vào.

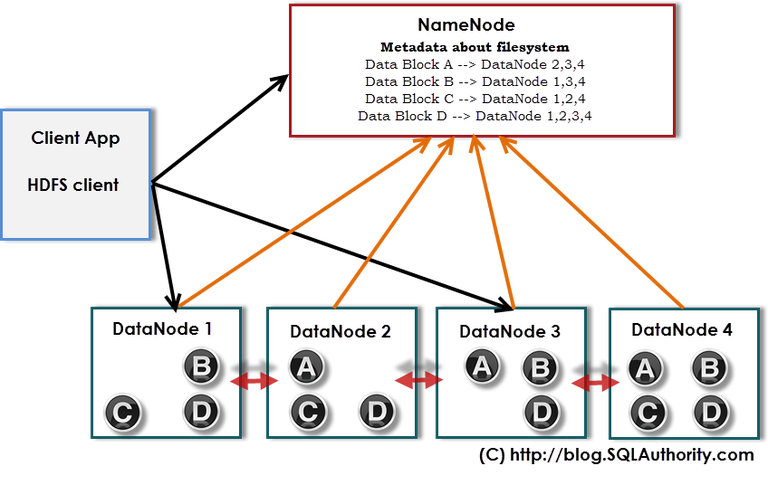
* Giai đoạn 2: Hadoop job client submit job (file jar, file thực thi) và các thiết lập cho JobTracker. Sau đó, master sẽ phân phối tác vụ đến các máy slave để theo dõi và quản lý tiến trình các máy này, đồng thời cung cấp thông tin về tình trạng và chẩn đoán liên quan đến job-client.
* Giai đoạn 3: TaskTrackers trên các node khác nhau thực thi tác vụ MapReduce và trả về kết quả output được lưu trong hệ thống file.
* Có hai loại node điều kiển quá trình thực hiện công việc (job): một jobtracker và một số tasktracker.
* Jobtracker kết hợp tất cả các công việc trên hệ thống bằng cách lập lịch công việc chạy trên các tasktracker. Chỉ có một JobTracker trên một cụm Hadoop. Nó thường chạy trên một máy chủ như là một nút master của cluster.
* Tasktracker chạy các nhiệm vụ (task) và gửi báo cáo thực hiện cho jobtracker, cái lưu giữ các bản ghi về quá trình xử lý tổng thể cho mỗi công việc (job). Mặc dù có một TaskTracker duy nhất cho một node slave, mỗi TaskTracker có thể sinh ra nhiều JVM để xử lý các nhiệm vụ Map hoặc Reduce song song. Nếu JobTracker không nhận được phản hồi từ một TaskTracker trong vòng một lượng thời gian đã quy định, nó sẽ cho rằng TaskTracker đã bị treo (cashed) và sẽ gửi lại nhiệm vụ tương ứng cho các nút khác trong cluster.

Hadoop chia đầu vào cho mỗi công việc MapReduce vào các mảnh (piece) có kích thước cố định gọi là các input split hoặc là các split. Hadoop tạo ra một task map cho mỗi split, cái chạy mỗi nhiệm vụ map do người sử dụng định nghĩa cho mỗi bản ghi (record) trong split. Đối với hầu hết công việc, kích thước split tốt nhất thường là kích thước của một block của HDFS, mặc định là 64MB, mặc dù nó có thể thay đổi được cho mỗi cluster (cho tất cả các file mới được tạo ra) hoặc định rõ khi mỗi file được tạo ra....

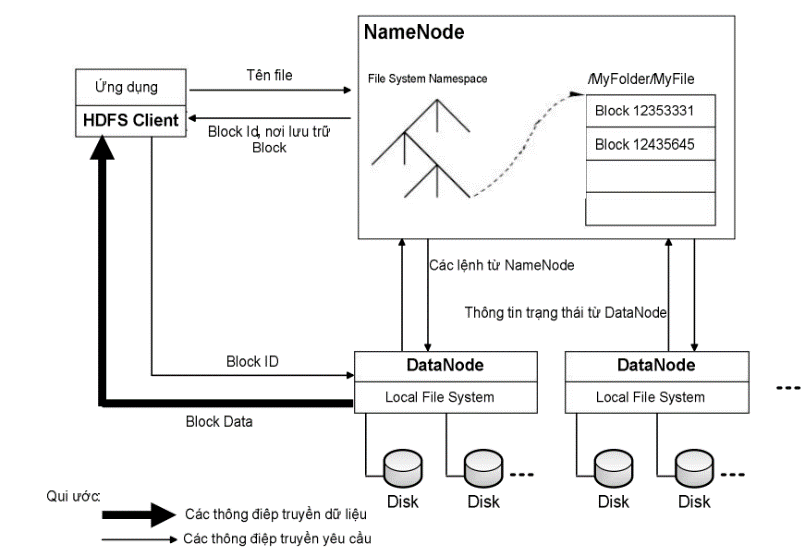
# HDFS

* HDFS tạo các mảnh nhỏ hơn của dữ liệu lớn và phân tán nó trên các nodes khác nhau. Nó cũng sao chép mỗi miếng dữ liệu nhỏ hơn nhiều lần trên nhiều nodes khác nhau. Do đó khi bất kỳ node nào có dữ liệu lỗi, hệ thống tự động dùng dữ liệu từ 1 node khác và tiếp tục xử lý.





* HDFS có kiến trúc master/slave. Trên một cluster chạy HDFS, có hai loại node là Namenode và Datanode. Một cluster có duy nhất một Namenode và có một hay nhiều Datanode.
  + - * Namenode đóng vai trò là master, quản lý hệ thống tập tin cũng như điều chỉnh truy cập đến các tập tin khác nhau. Tác vụ chính của NameNode là mở, đóng và đổi tên các tập tin, và thư mục và điều chỉnh truy cập đến hệ thống tập tin. Chỉ có 1 NameNode duy nhất trong kiến trúc. Nếu node này gặp lỗi thì toàn bộ ứng dụng Hadoop sẽ ngừng vì node này là nơi lưu tất cả siêu dữ liệu. Node này cực kỳ quan trọng, nó thường được nhân rộng sang các cụm khác.
      * Luôn có 1 DataNode cho mỗi máy chủ dữ liệu. Trong HDFS, 1 tập tin lớn được chia thành 1 hoặc nhiều khối và các khối này được lưu trong 1 tập các DataNodes. Tác vụ chính của DataNode là đọc và ghi vào hệ thống tập tin. DataNode cũng làm nhiệm vụ tạo, xóa, hoặc nhân rộng dữ liệu dựa trên chỉ dẫn từ NameNode. Nhiều DataNode chứa cùng các khối dữ liệu trong trường hợp bất kỳ DataNode nào gặp thảm họa, toàn bộ xử lý sẽ vẫn tiếp tục, DataNode khác sẽ thay thế vai trò để xử lý khối dữ liệu cụ thể của node bị lỗi. Hệ thống này cung cấp tính chịu lỗi rất cao để đối phó với thảm họa và cung cấp tính sẵn sàng cao.
* Đọc file và Ghi file trên HDFS:[[3]](#footnote-4)

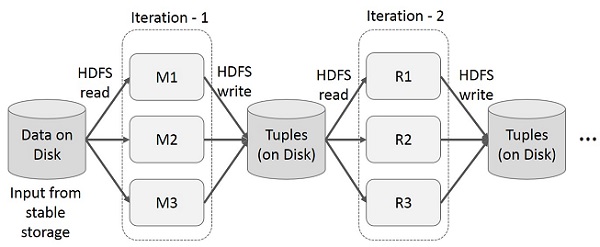


# RDD (Resilient Distributed DataSet)

* Cấu trúc dữ liệu: Bộ dữ liệu phân tán của các item được phân tán trên các node của cluster và có thể được xử lý song song. RDD có thể chứa bất kì kiểu dữ liệu nào của Python, Java, hoặc đối tượng Scala, bao gồm các kiểu dữ liệu do người dùng định nghĩa.
* Đặc điểm:
  + - * Resilient (Khả năng phục hồi): Tạo ra RDD mới thay vì sửa đổi RDD cũ. Tính chất này cho phép Spark xây dựng lại RDD từ RDD trước đó mà không phải chạy lại toàn bộ chương trình nếu gặp sự cố. -> điểm khác biệt so với DSM (Distributed Shared Memory: cho phép đọc ghi vào bộ nhớ) -> RDD hạn chế với các ứng dụng thực hiện ghi số lượng lớn nhưng chịu lỗi hiệu quả hơn, DSM khó sao lưu vì các bản sao của một tác vụ sẽ đọc/ghi vào cùng 1 địa chỉ bộ nhớ.
      * Distributed (Phân tán): Dữ liệu được phân bố ở nhiều nút trên 1 cụm.
      * Dataset: Mỗi dataset trong RDD được chia thành nhiều phân vùng logical, có thể được tính toán trên các node khác nhau của một cụm máy chủ (cluster).
* RDDs hỗ trợ hai kiểu thao tác thao tác: transformations và action.
  + - * Thao tác transformation: tạo RDD mới từ RDD có sẵn. Thao tác này ghi nhớ các bước thực hiện, chỉ được thực hiện khi trong quá trình có Action.

VD: map. flatMap, filter, reduceByKey, join, cogroup

* + - * Thao tác actions trả về giá trị cho chương trình điều khiển (driver program) sau khi thực hiện tính toán trên RDD cuối cùng.



* Một RDD phụ thuộc vào:
  + - * RDD trước đó sinh ra nó
      * Mảng các phân vùng dữ liệu
      * Hàm tính toán để thực hiện tính toán trên các phân vùng
      * Trình phân vùng
      * Locality info: phân vùng lưu trữ các bản ghi gần nhất
* RDD là một cấu trúc chứa các hướng dẫn về cách cụ thể hóa dữ liệu phân tán lớn và cách phân chia dữ liệu thành các phân vùng để Spark (sử dụng executor) xử lý một số trong đó. Spark thực hiện các công việc song song và RDD chia các phân vùng để xử lý song song. Bên trong mỗi phân vùng được xử lý tuần tự.
* Một số RDD phổ biến:[[4]](#footnote-5)
  + - * ParallelCollectionRDD
      * CoGroupedRDD
      * HadoopRDD
      * MapPartitionsRDD
      * CoalescedRDD
      * ShuffledRDD
      * ….
* Kể từ Spark 2.0, RDD được thay thế bằng Dataset

|  |  |  |
| --- | --- | --- |
| RDD | DataFrame | DataSet |
| Dữ liệu không cấu trúc: Tập hợp các phần tử các dữ liệu phân tán được đặt trên nhiều máy trong cluster.  Low-level API  Hiệu năng kém nhất | Dữ liệu có/bán cấu trúc: Tập hợp các dữ liệu phân tán được sắp xếp thành các cột được đặt tên  Hìgh-level API  Hiệu năng tốt nhất | Dữ liệu có/bán cấu trúc: giao diện hướng đối tượng  Hìgh-level API  An toàn đối với dữ liệu |

# Apache Spark

* Một open source cluster computing framework, cho phép xây dựng các mô hình dự đoán nhanh chóng với việc tính toán được thực hiện trên một nhóm các máy tính, có có thể tính toán cùng lúc trên toàn bộ tập dữ liệu mà không cần phải trích xuất mẫu tính toán thử nghiệm.
* Tính năng chính: Tính toán cụm trong bộ nhớ giúp tăng tốc độ xử lý của ứng dụng.
* Thành phần: gồm 5 phần chính:
  + - * Spark Core

Spark Core là nền tảng cho các thành phần còn lại và các thành phần này muốn khởi chạy được thì đều phải thông qua Spark Core do Spark Core đảm nhận vai trò thực hiện công việc tính toán và xử lý trong bộ nhớ (In-memory computing) đồng thời nó cũng tham chiếu các dữ liệu được lưu trữ tại các hệ thống lưu trữ bên ngoài.

* + - * Spark Streaming

Spark Streaming được sử dụng để thực hiện việc phân tích stream bằng việc coi stream là các mini-batches và thực hiệc kỹ thuật RDD transformation đối với các dữ liệu mini-batches này. Qua đó cho phép các đoạn code được viết cho xử lý batch có thể được tận dụng lại vào trong việc xử lý stream, làm cho việc phát triển lambda architecture được dễ dàng hơn.

* + - * Spark SQL

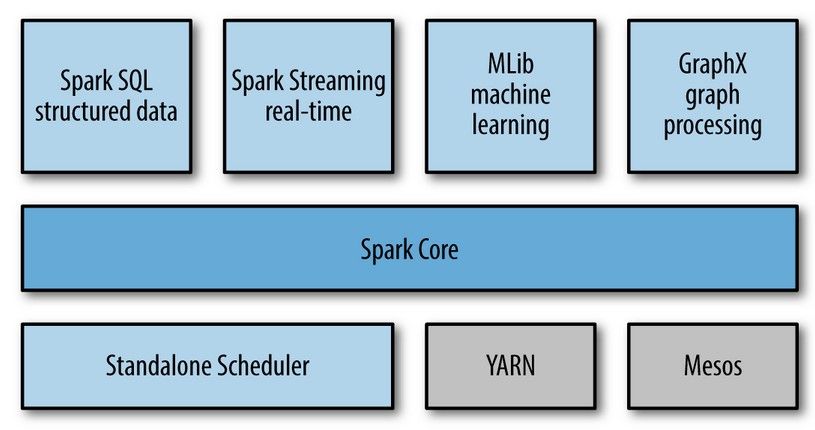
Spark SQL cung cấp kiểu data abstraction mới (SchemaRDD) hỗ trợ cho cả kiểu dữ liệu có cấu trúc (structured data) và dữ liệu nửa cấu trúc (semi-structured data). Spark SQL hỗ trợ DSL (Domain-specific language) để thực hiện các thao tác trên DataFrames bằng ngôn ngữ Scala, Java hoặc Python và nó cũng hỗ trợ cả ngôn ngữ SQL với giao diện command-line và ODBC/JDBC server.

* + - * MLlib

Một nền tảng học máy phân tán trên Spark cung cấp các thuật toán, mô hình thống kê và học máy có thể mở rộng được.

* + - * GraphX

Nền tảng xử lý đồ thị trên Spark cho phép người dùng tương tác xây dựng, biến đổi về dữ liệu có cấu trúc dạng đồ thị theo quy mô.

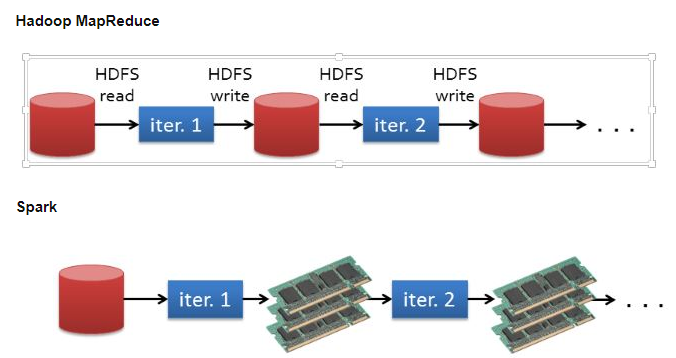


Cấu trúc Apache Spark

* Spark sử dụng API Resilient Distributed Dataset (RDD) để xử lý dữ liệu.
  + - * Trong Hadoop, hầu hết thời gian dành cho các thao tác đọc ghi HDFS -> RDD: hỗ trợ tính toán xử lý trong bộ nhớ.
      * Lưu trữ kết quả trung gian trong bộ nhớ phân tán thay vì lưu trữ trong đĩa.



* + - * Nếu các truy vấn khác nhau được chạy trên cùng một tập hợp dữ liệu, dữ liệu cụ thể này có thể được lưu trong bộ nhớ để có thời gian thực hiện tốt hơn.
* Hữu ích cho cả học máy và cả xử lý luồng dữ liệu hoàn chỉnh.
* So sánh với Hadoop:
  + - * Một trong những lý do khiến Spark chạy nhanh hơn Hadoop MapReduce đó là ở mỗi tác vụ dữ liệu được nạp lên bộ nhớ và xử lý ở đó, những tác vụ sau có thể sử dụng dữ liệu nằm trên bộ nhớ thay vì phải đọc ghi liên tục vào HDFS như Hadoop MapReduce.



* + - * Spark: có thể xử lý realtime với tốc độ hàng triệu sự kiện/s (hỗ trợ truyền dữ liệu cùng với phân tán so với MapReduce là xử lý hàng loạt kiêm phân tán trên một lượng lớn dữ liệu). Nếu muốn thực hiện một công việc lặp lại, phải kết hợp một chuỗi công việc MapReduce và thực hiện theo trình tự, job sau phụ thuộc vào việc hoàn thành job trước trong khi Spark cho phép sử dụng các pipeline đa bước, chia sẻ dữ liệu trong bộ nhớ mà không cần vào đĩa qua mô hình DAG (directed acyclic graph).
      * Spark có thể chạy trên HDFS để tận dụng bộ nhớ replicated phân tán, sử dụng cùng MapReduce (trong đó, MapReduce: xử lý hàng loạt, Spark: xử lý realtime), Spark cũng có thể chạy hoàn toàn tách biệt với Hadoop, tích hợp với các trình quản lý cụm thay thế như Mesos và các nền tảng lưu trữ thay thế như Cassandra và Amazon S3.
      * Spark tích hợp Hadoop:

+ Trình quản lý tài nguyên YARN: lên lịch các task trên các node có sẵn trong cluster.

+ HDFS: lưu trữ dữ liệu khi cluster hết bộ nhớ trống, lưu trữ lịch sử dữ liệu khi Spark không chạy

+ Khôi phục dữ liệu khi các node riêng lẻ bị lỗi

+ Bảo mật dữ liệu: quan trọng khi Spark được sử dụng trong các ngành như tài chính, y tế

+ Nền tảng dữ liệu phân tán: các jobs của Spark có thể được triển khai trên các tài nguyên có sẵn ở 1 vị trí bất kì trong cluster phân tán mà không cần phân bổ và theo dõi thủ công các jobs riêng lẻ.

* Một số khái niệm chung:
  + - * Jobs: Các mục công việc cấp cao nhất được gửi đến bộ lập lịch. Mỗi job có thể có nhiều giai đoạn trung gian.
      * Tasks: đơn vị công việc, mỗi đơn vị gửi đến 1 máy.
      * Stages: tập hợp các nhiệm vụ trung gian cho job, trong đó các task tính cùng chức năng cùng trên phân vùng của RDD. Gồm 2 loại: Result Stage và ShuffleMap Stage
      * Cache Tracking: Bộ lập lịch DAG chỉ ra RDD nào được lưu trong bộ nhớ để tránh tính toán lại và ghi nhớ các Shuffle Map Stages đã tạo các tệp đầu ra để tránh tính lại map side của một lần xáo
      * Preferred locations: Bộ lập lịch DAG tính toán nơi chạy từng task
* API
  + - * DataFrame
      * SparkStreaming

Spark có khả năng tích hợp chặt chẽ giữa khả năng xử lý luồng và xử lý hàng loatj. Spark Streaming hỗ trợ nhập dữ liệu từ nhiều nguồn dữ liệu, bao gồm các luồng trực tiếp như Twitter, cảm biến,… Dữ liệu được xử lý rồi có thể gửi đến các dịch vụ lưu trữ như HDFS, AWS S3 hoặc được sử dụng trực tiếp. Về mặt logic, Spark Streaming thể hiện đầu vào là một dạng dữ liệu liên tục dưới dạng luồng rời rạc hoặc DStream, xử lý như một chuỗi RDDs(một batch gồm 1 chuỗi các sự kiện được nhóm lại trong 1 khoảng thời gian 500-5000ms, khung thời gian càng ngắn số lượng RDDs được tạo ra càng lớn), cho phép khả năng xử lý hàng loạt.

* Ví dụ
  + - * Xây dựng hồ sơ người dùng từ một dịch vụ âm nhạc.
* Các cách triển khai Spark:
  + - * Spark Standalone: Spark sử dụng HDFS làm nền và được phân bổ rõ ràng. Spark và MapReduce chạy song song để thực hiện tất cả các jobs trên cluster
      * Hadoop Yarn:
      * Spark in MapReduce (SIMR):
* VD: film recommendation:
  + - * Tìm mô hình gợi ý với Spark
      * Tải và xử lý tập dữ liệu với Spark

+ Tải data vào DataFrames

+

* + - * Sử dụng thuật toán ALS để tạo gợi ý phim
      * Kiểm tra độ chính xác

1. <https://kipalog.com/posts/Co-ban-ve-Hadoop> [↑](#footnote-ref-2)
2. <https://kipalog.com/posts/Tong-quan-mo-hinh-lap-trinh-MapReduce> [↑](#footnote-ref-3)
3. <https://techblog.vn/su-khac-nhau-ve-hdfs-va-hdfs2> [↑](#footnote-ref-4)
4. <https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-rdd.html> [↑](#footnote-ref-5)