

# Apache Spark RDD

## Resilient Distributed Datasets

Resilient Distributed Datasets (RDD) là một cấu trúc dữ liệu cơ bản của Spark. Nó là một tập hợp bất biến phân tán của một đối tượng. Mỗi dataset trong RDD được chia ra thành nhiều phần vùng logical. Có thể được tính toán trên các node khác nhau của một cụm máy chủ (cluster).

RDDs có thể chứa bất kỳ kiểu dữ liệu nào của Python, Java, hoặc đối tượng Scala, bao gồm các kiểu dữ liệu do người dùng định nghĩa. Thông thường, RDD chỉ cho phép đọc, phân mục tập hợp của các bản ghi. RDDs có thể được tạo ra qua điều khiển xác định trên dữ liệu trong bộ nhớ hoặc RDDs, RDD là một tập hợp có khả năng chịu lỗi mỗi thành phần có thể được tính toán song song.

Có hai cách để tạo RDDs:

- Tạo từ một tập hợp dữ liệu có sẵn trong ngôn ngữ sử dụng như Java, Python, Scala.
- Lấy từ dataset hệ thống lưu trữ bên ngoài như HDFS, Hbase hoặc các cơ sở dữ liệu quan hệ.

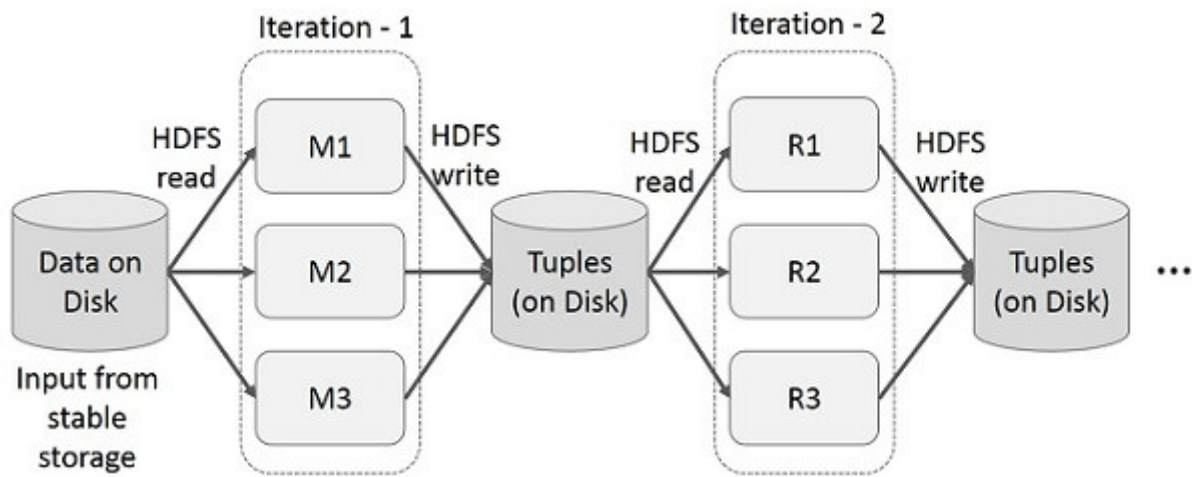
## Thực thi trên MapReduce

MapReduce được áp dụng rộng rãi để xử lý và tạo các bộ dữ liệu lớn với thuật toán xử lý phân tán song song trên một cụm. Nó cho phép người dùng viết các tính toán song song, sử dụng một tập hợp các toán tử cấp cao, mà không phải lo lắng về xử lý/phân phối công việc và khả năng chịu lỗi.

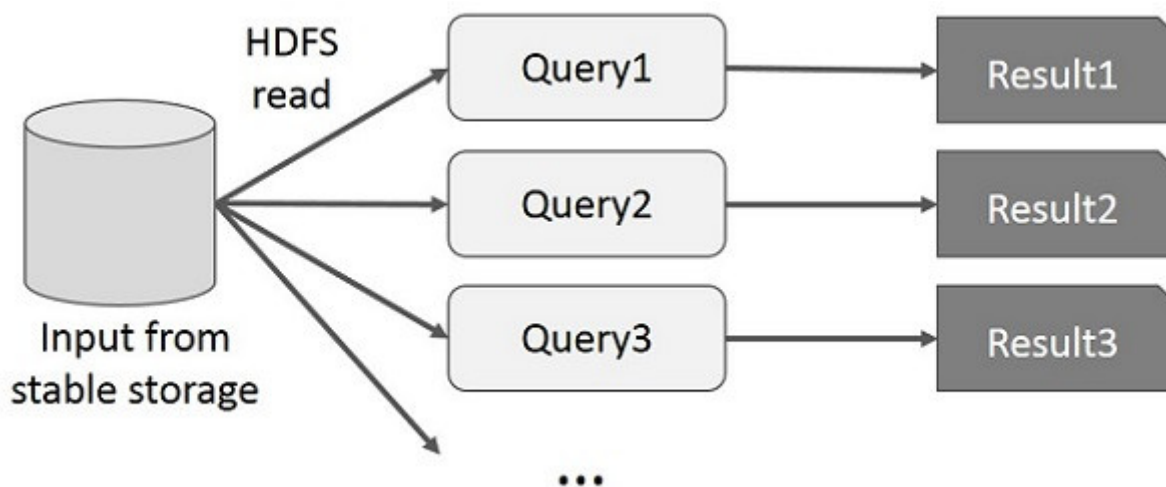
Tuy nhiên, trong hầu hết các framework hiện tại, cách duy nhất để sử dụng lại dữ liệu giữa các tính toán (Ví dụ: giữa hai công việc MapReduce) là ghi nó vào storage (Ví dụ: HDFS). Mặc dù framework này cung cấp nhiều hàm thư viện để truy cập vào tài nguyên tính toán của cụm Cluster, điều đó vẫn là chưa đủ.

Cả hai ứng dụng Lặp (Iterative) và Tương tác (Interactive) đều yêu cầu chia sẻ truy cập và xử lý dữ liệu nhanh hơn trên các công việc song song. Chia sẻ dữ liệu chậm trong MapReduce do sao chép tuần tự và tốc độ I/O của ổ đĩa. Về hệ thống lưu trữ, hầu hết các ứng dụng Hadoop, cần dành hơn 90% thời gian để thực hiện các thao tác đọc-ghi HDFS.

### - Iterative Operation trên MapReduce:



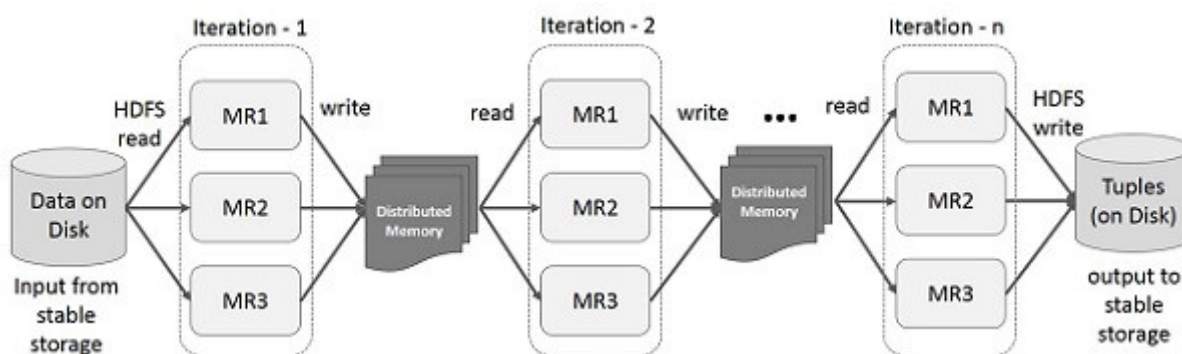
### - Interactive Operations trên MapReduce:



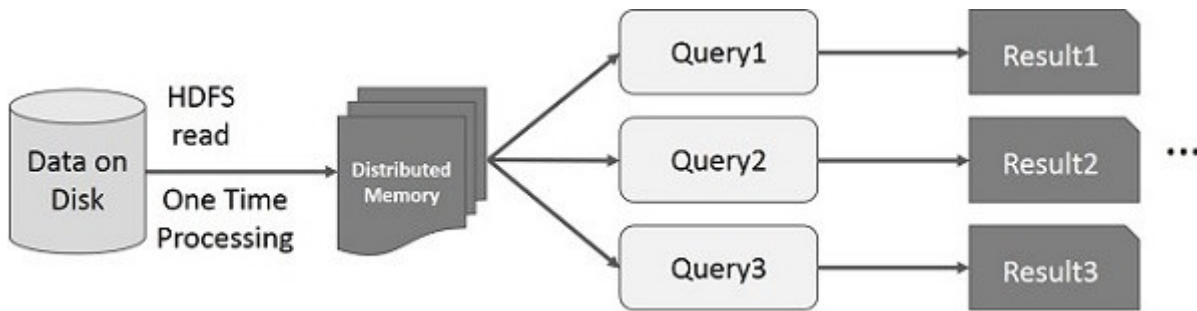
## Thực thi trên Spark RDD

Để khắc phục được vấn đề về MapReduce, các nhà nghiên cứu đã phát triển một framework chuyên biệt gọi là Apache Spark. Ý tưởng chính của Spark là Resilient Distributed Datasets (RDD); nó hỗ trợ tính toán xử lý trong bộ nhớ. Điều này có nghĩa, nó lưu trữ trạng thái của bộ nhớ dưới dạng một đối tượng trên các công việc và đối tượng có thể chia sẻ giữa các công việc đó. Việc xử lý dữ liệu trong bộ nhớ nhanh hơn 10 đến 100 lần so với network và disk.

### - Iterative Operation trên Spark RDD:

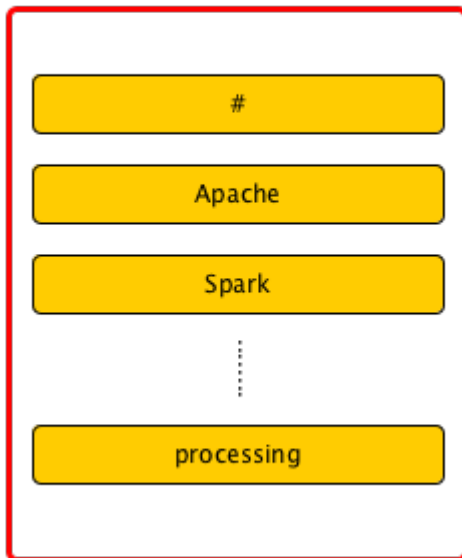


## - Interactive Operations trên Spark RDD:

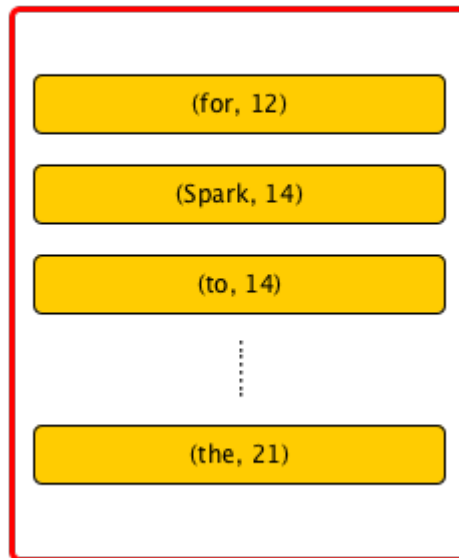


## Các loại RDD

### RDD of Strings



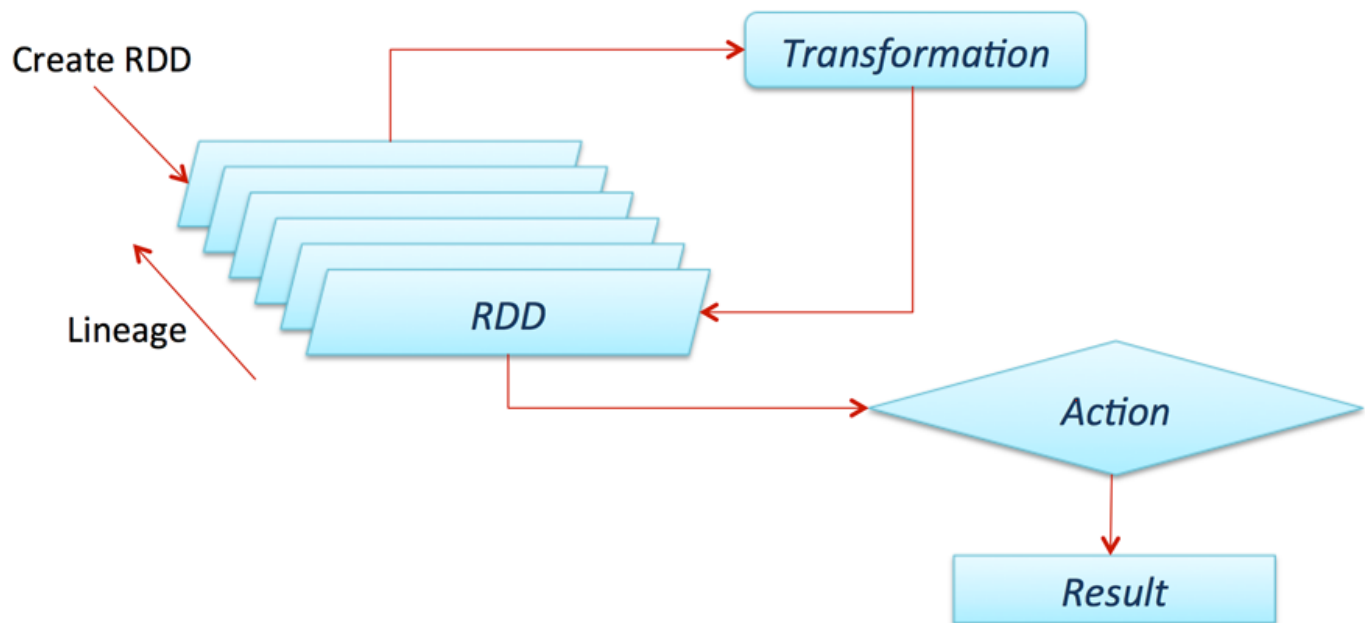
### RDD of Pairs



- Các RDD biểu diễn một tập hợp cố định, đã được phân vùng các record để có thể xử lý song song.
- Các record trong RDD có thể là đối tượng Java, Scala hay Python tùy lập trình viên chọn. Không giống như DataFrame, mỗi record của DataFrame phải là một dòng có cấu trúc chứa các field đã được định nghĩa sẵn.
- RDD đã từng là API chính được sử dụng trong series Spark 1.x và vẫn có thể sử dụng trong version 2.X nhưng không còn được dùng thường xuyên nữa.
- RDD API có thể được sử dụng trong Python, Scala hay Java:
  - Scala và Java: Performance tương đương trên hầu hết mọi phần. (Chi phí lớn nhất là khi xử lý các raw object)
  - Python: Mất một lượng performance, chủ yếu là cho việc serialization giữa tiến trình Python và JVM

## Các transformation và action với RDD

RDD cung cấp các transformation và action hoạt động giống như DataFrame lẫn DataSets. Transformation xử lý các thao tác lazily và Action xử lý thao tác cần xử lý tức thời.



### - Một số transformation:

Nhiều phiên bản transformation của RDD có thể hoạt động trên các Structured API, `transformation` xử lý lazily, tức là chỉ giúp dựng execution plans, dữ liệu chỉ được truy xuất thực sự khi thực hiện `action`

- **distinct**: loại bỏ trùng lặp trong RDD
- **filter**: tương đương với việc sử dụng where trong SQL – tìm các record trong RDD xem những phần tử nào thỏa điều kiện. Có thể cung cấp một hàm phức tạp sử dụng để filter các record cần thiết – Như trong Python, ta có thể sử dụng hàm lambda để truyền vào filter
- **map**: thực hiện một công việc nào đó trên toàn bộ RDD. Trong Python sử dụng lambda với từng phần tử để truyền vào map
- **flatMap**: cung cấp một hàm đơn giản hơn hàm map. Yêu cầu output của map phải là một structure có thể lặp và mở rộng được.
- **sortBy**: mô tả một hàm để trích xuất dữ liệu từ các object của RDD và thực hiện sort được từ đó.
- **randomSplit**: nhận một mảng trọng số và tạo một random seed, tách các RDD thành một mảng các RDD có số lượng chia theo trọng số.

### - Một số action:

Action thực thi ngay các transformation đã được thiết lập để thu thập dữ liệu về driver để xử lý hoặc ghi dữ liệu xuống các công cụ lưu trữ.

- **reduce**: thực hiện hàm reduce trên RDD để thu về 1 giá trị duy nhất
- **count**: đếm số dòng trong RDD
- **countApprox**: phiên bản đếm xấp xỉ của count, nhưng phải cung cấp timeout vì có thể không nhận được kết quả.

- **countByValue**: đếm số giá trị của RDD  
chỉ sử dụng nếu map kết quả nhỏ vì tất cả dữ liệu sẽ được load lên memory của driver để tính toán  
chỉ nên sử dụng trong tình huống số dòng nhỏ và số lượng item khác nhau cũng nhỏ.
- **countApproxDistinct**: đếm xấp xỉ các giá trị khác nhau
- **countByValueApprox**: đếm xấp xỉ các giá trị
- **first**: lấy giá trị đầu tiên của dataset
- **max và min**: lần lượt lấy giá trị lớn nhất và nhỏ nhất của dataset
- **take và các method tương tự**: lấy một lượng giá trị từ trong RDD. take sẽ trước hết scan qua một partition và sử dụng kết quả để dự đoán số lượng partition cần phải lấy thêm để thỏa mãn số lượng lấy.
- **top và takeOrdered**: top sẽ hiệu quả hơn takeOrdered vì top lấy các giá trị đầu tiên được sắp xếp ngầm trong RDD.
- **takeSamples**: lấy một lượng giá trị ngẫu nhiên trong RDD

## Một số kỹ thuật đối với RDD

- Lưu trữ file:

- Thực hiện ghi vào các file plain-text
- Có thể sử dụng các codec nén từ thư viện của Hadoop
- Lưu trữ vào các database bên ngoài yêu cầu ta phải lặp qua tất cả partition của RDD – Công việc được thực hiện ngầm trong các high-level API
- sequenceFile là một flat file chứa các cặp key-value, thường được sử dụng làm định dạng input/output của MapReduce. Spark có thể ghi các sequenceFile bằng cách ghi lại các cặp key-value
- Đồng thời, Spark cũng hỗ trợ ghi nhiều định dạng file khác nhau, cho phép define các class, định dạng output, config và compression scheme của Hadoop.

- Caching: Tăng tốc xử lý bằng cache

- Caching với RDD, Dataset hay DataFrame có nguyên lý như nhau.
- Chúng ta có thể lựa chọn cache hay persist một RDD, và mặc định, chỉ xử lý dữ liệu trong bộ nhớ

- Checkpointing: Lưu trữ lại các bước xử lý để phục hồi

- Checkpointing lưu RDD vào đĩa cứng để các tiến trình khác để thể sử dụng lại RDD point này làm partition trung gian thay vì tính toán lại RDD từ các nguồn dữ liệu gốc
- Checkpointing cũng tương tự như cache, chỉ khác nhau là lưu trữ vào đĩa cứng và không dùng được trong API của DataFrame
- Cần sử dụng nhiều để tối ưu tính toán.