Xử lý dữ liệu Lớn

Tìm hiểu Spark

Người thực hiện: Nguyễn Tuấn Anh

MSSV: 51800840

# Spark properties

Thuộc tính Spark kiểm soát hầu hết các cài đặt ứng dụng và được cấu hình riêng cho từng ứng dụng,

Các thuộc tính này có thể được đặt trực tiếp trên SparkConf được chuyển đến SparkContext.SparkConf cho phép cài đặt cấu hình một số thuộc tính chung.

Về pyspark

Các lớp công khai:

* SparkContext: Điểm vào chính cho chức năng Spark
* RDD: Tập dữ liệu phân tán có khả năng phục hồi, phần trừu tượng cơ bản trong Spark.
* Broadcast: Một biến quảng bá được sử dụng lại trên các tác vụ.
* Accumulator: Một biến chia sẻ mà các tác vụ chỉ có thể thêm giá trị vào.
* SparkConf: Để cấu hình Spark
* SparkFiles: Truy cập các tệp được gửi cùng với công việc
* StorageLevel: Mức độ bền bỉ của bộ nhớ cache tinh tế hơn.
* TaskContext: Thông tin về nhiệm vụ đang chạy hiện tại, có sẵn trên công nhân và thử nghiệm.
* RDDBarrier: Bao bọc một RDD dưới một giai đoạn rào cản để thực hiện rào cản.
* BarrierTaskContext: cung cấp thêm thông tin và công cụ để thực hiện rào cản.
* BarrierTaskInfo: thông tin về nhiệm vụ rào cản.

# Spark RDD

RDD (Resilient Distributed Dataset) hay còn gọi là tập dữ liệu phân tán đàn hồi. Là tập dữ liệu cơ bản của Apache Spark, là một tập hợp bất biến của các đối tượng tính toán trên các nút khác nhau của cụm.

Ý nghĩa tên RDD:

* Có khả năng phục hồi, tức là có khả năng chịu lỗi với sự trợ giúp của đồ thị dòng RDD và do đó có thể tính toán lại các phân vùng bị thiếu hoặc bị hỏng do lỗi nút.
* Phân tán vì dữ liệu nằm trên nhiều nút
* Tập dữ liệu đại diện cho các bản ghi dữ liệu làm việc. Người dùng có thể tái tập dữ liệu bên ngoài, có thể là tệp JSon, tệp CSV, tệp văn bản hoặc cơ sở dữ liệu thông qua JDBC không có cấu trúc cụ thể.

Có 4 chức năng chính:

* Bộ sưu tập dữ liệu phân tán
* Khả năng chịu lỗi
* Hoạt động song song phân chia
* Khả năng sử dụng nhiều nguồn dữ liệu

Có hai cách để tạo RDDs:

* Song song một tập hợp hiện có trong chương trình điều khiển.
* Tham chiếu tập dữ liệu trong hệ thống lưu trữ bên ngoài, chẳng hạn như hệ thống tệp chia sẻ, HDFS, HBase hoặc bất kỳ nguồn dữ liệu nào cung cấp Định dạng đầu vào Hadoop.

Spark sử dụng khái niệm RDD để đạt được các hoạt động MapReduce nhanh hơn và hiệu quả hơn.

Lập trình viên cũng có thể gọi một phương thức duy trì để chỉ ra RDD mà họ muốn sử dụng lại trong các hoạt động sau này. Spark giữ các RDD liên tục trong bộ nhớ theo mặc định, nhưng nó có thể tràn chúng ra đĩa nếu không có đủ RAM. Người dùng cũng có thể yêu cầu các chiến lược duy trì khác, chẳng hạn như chỉ lưu trữ RDD trên đĩa hoặc sao chép nó trên các máy, thông qua các cờ để duy trì.

## Mapreduce vs Spark RDD

### 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.

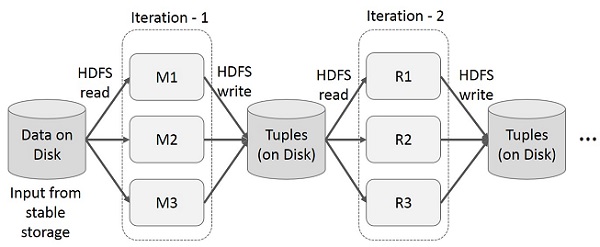


Figure 1 Hoạt động lặp lại trên Mapreduce

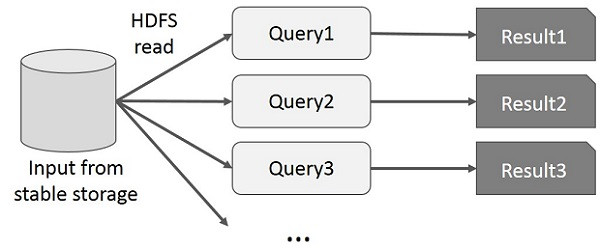


Figure 2 Hoạt động tương tác trên Mapreduce

### Thực thi trên Spark RDD

Để khắc phục được vấn đề về MapRedure, 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.

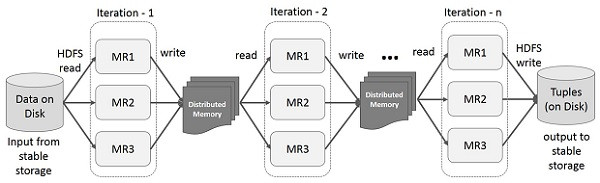


Figure 3 Hoạt động lặp lại trên RDD

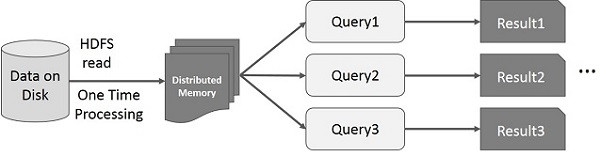
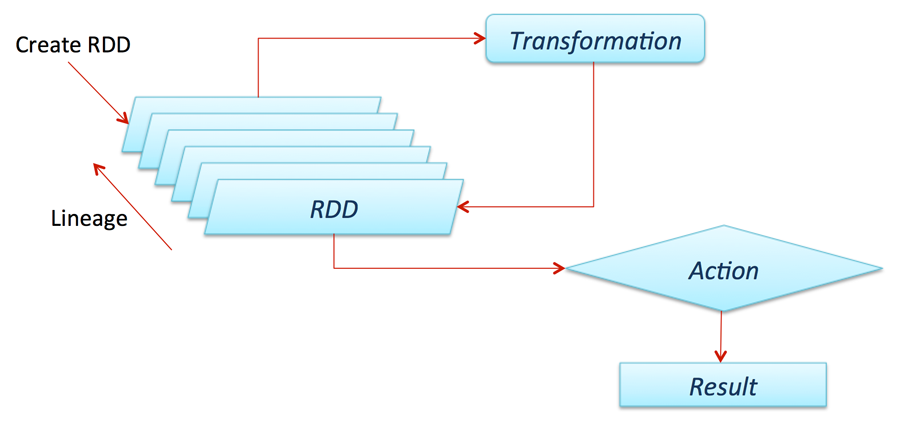


Figure 4 Hoạt động tương tác trên RDD

## 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 lắp 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

# Spark DataFrame

Trong Spark, DataFrame là một tập hợp dữ liệu phân tán được tổ chức thành các cột được đặt tên. Về mặt khái niệm, nó tương đương với một bảng trong cơ sở dữ liệu quan hệ hoặc một khung dữ liệu trong R / Python, nhưng với các tối ưu hóa phong phú hơn. DataFrames có thể được xây dựng từ nhiều nguồn như: tệp dữ liệu có cấu trúc, bảng trong Hive, cơ sở dữ liệu bên ngoài hoặc RDD hiện có.

## Tạo DataFrames



## Tạo dataFrames từ danh sách các hàng

