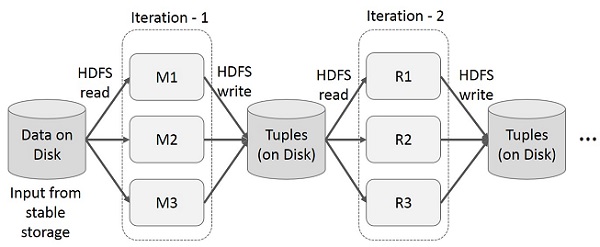
# Báo cáo tháng 9

## Tìm hiểu về Spark

## Ưu điểm của Spark so với Hadoop

1. Iterative Operations on MapReduce

Kết quả của các Iterative Operation sẽ được lưu trữ bởi HDFS trên Disk. Khi một Iteration khác cần kết quả đó làm input thì phải đọc từ disk vào, gây ra chi phí đáng kể do sao chép dữ liệu, disk I/O và tuần tự hóa, làm cho hệ thống chậm.



Hình 1. Iterative Operations on MapReduce

1. Iterative Operations on Spark RDD

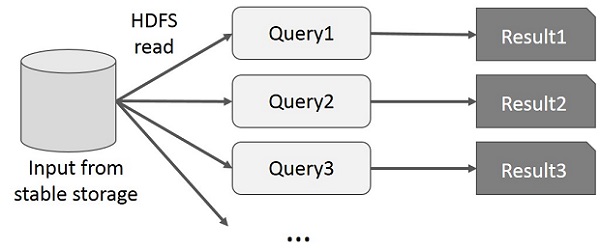
Kết quả trung gian được lưu trữ trong Distriuted Memmory (RAM) thay cho Stable Storage (Disk) và làm cho hệ thống nhanh hơn. Nếu RAM không đủ thì nó sẽ lưu trữ kết quả trung gian đó trên Disk.



Hình 2. Iterative Operations on Spark RDD

1. Interactive Operations on MapReduce

Các truy vấn trên cùng một tập dữ liệu khiến hoạt động Disk I/O lặp lại nhiều lần, chiếm phần lớn thời gian thực thi của ứng dụng.



Hình 3. Interactive Operations on MapReduce

1. Interactive Operaions on Spark RDD

Nếu các truy vấn khác nhau được chạy trên cùng một tập dữ liệu nhiều lần, dữ liệu cụ thể này có thể được lưu trong RAM để có thời gian thực hiện tốt hơn.



Hình 4. Interactive Operations on Spark RDD

## RDD, Dataframe và Dataset

1. RDD
2. Dataframe
3. Dataset

## Lập trình với thư viện Pyspark trên Python

1. So sánh các phương pháp tìm max

|  |  |
| --- | --- |
| Code | Time (s) |
| df.groupBy().max('VolData','NoSMS','DurationVoice') | 33.30 |
| for f in features:  max\_values.append(df.groupBy().max(f)) | 98.17 |
| df.select([max(f) for f in features]) | 33.34 |
| df.groupBy().agg(max('VolData'),max('NoSMS'),  max('DurationVoice')) | 32.67 |
| spark.sql(  'SELECT MAX(VolData),MAX(NoSMS),MAX(DurationVoice)  FROM user') | 34.23 |
| max\_values = rdd.map(lambda x: [x[f] for f in features])  .reduce(lambda a, b:  [max(a[i], b[i]) for i in range(n\_features)]) | 137.69 |

1. Hàm persist()

Thường thì mỗi transformation sẽ thực hiện chạy lại mỗi khi bạn chạy actions. Tuy nhiên, bạn có thể persist 1 RDD trong bộ nhớ sử dụng phương thức *persist* (hoặc *caching*). Điều này rất tiện lợi cho việc tính toán ngay tại bộ nhớ trong và tái sử dụng chúng cho actions khác trên tập dữ liệu, nó làm cho việc tính toán thực hiện nhanh hơn và khả năng chịu lỗi tốt . Cụ thể, lần đầu tiên tính toán với 1 actions, nó sẽ lưu trữ tại bộ nhớ trong trên các node. Nếu 1 phần nào đó của RDD bị mất, nó sẽ tự động tính toán lại sử dụng transformation được tạo ban đầu.

Mỗi RDD được lưu trữ với các mức khác nhau. Ví dụ, bạn có thể persist dữ liệu trên ổ cứng, bộ nhớ trong, hoặc sao lặp qua các node. Các mức lưu trữ tùy thuộc truyền đối tượng org.apache.spark.storage.StorageLevel vào hàm *persist()*. Phương thức *cache()* là cách viết tắt cho phương thức *persist* với mức lưu trữ mặc định, cụ thể là *StorageLevel.MEMORY\_ONLY*.

Sau đây là các mức lưu trữ :

|  |  |
| --- | --- |
| Storage Level | Ý nghĩa |
| MEMORY\_ONLY | Lưu trữ RDD như đối tượng deserialized Java trên JVM. Nếu RDD vượt quá bộ nhớ trong thì 1 số vùng sẽ không được cached và nó sẽ tính toán lại mỗi khi nó cần. Đây là mức lưu trữ mặc định. |
| DISK\_ONLY | Lưu trữ các vùng RDD chỉ trên ổ cứng. |
| MEMORY\_ONLY\_2, MEMORY\_AND\_DISK\_2, ... | Giống như các mức trên nhưng sao lặp mỗi vùng RDD trên 2 cluster nodes. |

Để thay đổi mức lưu trữ, sử dụng phương thức *apply()*của đối tượng StorageLevel.

Bảng sau đây so sánh thời gian thực thi trong bài toán “Tính toán độ tương đồng giữa các movie” cho bộ dữ liệu MovieLens **ml-100k**:

|  |  |  |  |
| --- | --- | --- | --- |
| Number of movies | 1 | 20 | 50 |
| persist(MEMORY\_ONLY) = cache() | 23 | 42.5 | 73.5 |
| no persist | 34.6 | 386.8 | 904.6 |

All\_User0.csv : 6,513,721 : 259.83s – 71s

All: 65,523,580 : 5476.9s