**Xử lý dữ liệu lớn - 51603183**

Bài tập 2: Tìm hiểu Spark

1. **Spark Properties**

Spark properties 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 tới SparkContext. SparkConf cho phép chúng ta đặt cấu hình một số thuộc tính phổ biến (ví dụ: URL chính và tên ứng dụng), cũng như các cặp key-value tùy ý thông qua phương thức set (). Ví dụ: chúng ta có thể khởi tạo một ứng dụng với hai luồng như sau:

**val** conf **=** **new** **SparkConf**()

.setMaster("local[2]")

.setAppName("CountingSheep")

**val** sc **=** **new** **SparkContext**(conf)

Lưu ý rằng có thể có nhiều hơn 1 luồng ở chế độ local và trong những trường hợp như Spark Streaming, có thể yêu cầu nhiều hơn 1 luồng để ngăn chặn bất kỳ loại vấn đề thiếu dữ liệu nào. Các thuộc tính chỉ định một số khoảng thời gian nên được cấu hình với một đơn vị thời gian. Ví dụ như sau:

25ms (milliseconds)

5s (seconds)

10m or 10min (minutes)

3h (hours)

5d (days)

1y (years)

Thuộc tính chỉ định kích thước byte phải được cấu hình với đơn vị kích thước. Ví dụ như sau:

1b (bytes)

1k or 1kb (kibibytes = 1024 bytes)

1m or 1mb (mebibytes = 1024 kibibytes)

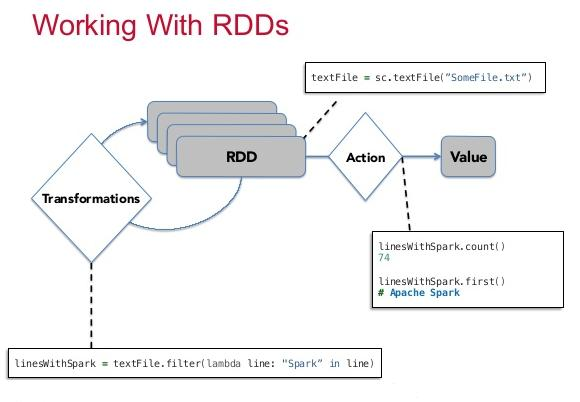
1g or 1gb (gibibytes = 1024 mebibytes)

1t or 1tb (tebibytes = 1024 gibibytes)

1p or 1pb (pebibytes = 1024 tebibytes)

Trong khi các số không có đơn vị thường được hiểu là byte, một số ít được hiểu là KiB hoặc MiB.

1. **Spark RDD**



Ở cấp độ cao, mọi ứng dụng Spark bao gồm một chương trình trình điều khiển chạy chức năng chính của người dùng và thực hiện các hoạt động song song khác nhau trên một cụm. Tính trừu tượng chính mà Spark cung cấp là tập dữ liệu phân tán có khả năng phục hồi (RDD), là tập hợp các phần tử được phân vùng trên các nút của cụm có thể hoạt động song song. RDD được tạo bằng cách bắt đầu với một tệp trong hệ thống tệp Hadoop (hoặc bất kỳ hệ thống tệp nào khác được Hadoop hỗ trợ) hoặc một bộ sưu tập Scala hiện có trong chương trình điều khiển và chuyển đổi nó. Người dùng cũng có thể yêu cầu Spark duy trì một RDD trong bộ nhớ, cho phép nó được sử dụng lại một cách hiệu quả trong các hoạt động song song. Cuối cùng, các RDD tự động phục hồi sau các lỗi của nút. Sự trừu tượng thứ hai trong Spark là các biến được chia sẻ có thể được sử dụng trong các hoạt động song song. Theo mặc định, khi Spark chạy song song một hàm dưới dạng một tập hợp các tác vụ trên các nút khác nhau, nó sẽ gửi một bản sao của từng biến được sử dụng trong hàm cho mỗi tác vụ. Đôi khi, một biến cần được chia sẻ giữa các tác vụ hoặc giữa các tác vụ và chương trình điều khiển. Spark hỗ trợ hai loại biến chia sẻ: biến quảng bá, có thể được sử dụng để lưu trữ một giá trị trong bộ nhớ trên tất cả các nút và bộ tích lũy, là những biến chỉ được “thêm vào”, chẳng hạn như bộ đếm và tổng.

Spark xoay quanh khái niệm về tập dữ liệu phân tán có khả năng phục hồi (RDD), là một tập hợp các phần tử có khả năng chịu lỗi và có thể hoạt động song song. Có hai cách để tạo RDD: song song một bộ sưu tập hiện có trong chương trình trình điều khiển hoặc tham chiếu tập dữ liệu trong hệ thống lưu trữ bên ngoài, chẳng hạn như hệ thống tệp được chia sẻ, HDFS, HBase hoặc bất kỳ nguồn dữ liệu nào cung cấp Hadoop InputFormat.

* 1. **Parallelized Collections**

Các tập hợp song song được tạo bằng cách gọi phương thức SparkContext’s parallelize trên tập hợp hiện có trong chương trình điều khiển. Các phần tử của bộ sưu tập được sao chép để tạo thành một tập dữ liệu phân tán có thể được vận hành song song. Ví dụ: đây là cách tạo một tập hợp song song chứa các số từ 1 đến 5:

**val** data **=** **Array**(1, 2, 3, 4, 5)

**val** distData **=** sc.parallelize(data)

Sau khi được tạo, tập dữ liệu phân tán (distData) có thể được vận hành song song. Ví dụ, chúng ta có thể gọi distData.reduce ((a, b) => a + b) để cộng các phần tử của mảng.

Một tham số quan trọng cho các bộ sưu tập song song là số lượng phân vùng để cắt tập dữ liệu. Spark sẽ chạy một tác vụ cho mỗi phân vùng của cụm. Thông thường, người dùng muốn 2-4 phân vùng cho mỗi CPU trong cụm của mình.Và Spark cố gắng đặt số lượng phân vùng tự động dựa trên cụm của người dùng. Tuy nhiên cũng có thể đặt theo cách thủ công bằng cách chuyển thành tham số thứ hai để song song hóa (ví dụ: sc.parallelize (data, 10)).

* 1. **External Datasets**

Spark có thể tạo tập dữ liệu phân tán từ bất kỳ nguồn lưu trữ nào được Hadoop hỗ trợ, bao gồm hệ thống local file, HDFS, Cassandra, HBase, Amazon S3, v.v. Spark hỗ trợ tệp văn bản, SequenceFiles và bất kỳ Hadoop InputFormat nào khác. Các RDD của tệp văn bản có thể được tạo bằng cách sử dụng phương thức textFile của SparkContext. Phương thức này lấy một URI cho tệp (đường dẫn cục bộ trên máy hoặc URI hdfs: //, s3a: //, v.v.) và đọc nó như một tập hợp các dòng. Ví dụ như sau:

scala> **val** distFile **=** sc.textFile("data.txt")

distFile**:** org.apache.spark.rdd.RDD[String] **=** data.txt **MapPartitionsRDD**[10] at textFile at <console**>:**26

Sau khi được tạo, distFile có thể được thực hiện bằng các hoạt động của tập dữ liệu. Ví dụ, chúng ta có thể cộng kích thước của tất cả các đường bằng cách sử dụng thao tác map và reduce như sau: distFile.map (s => s.length) .reduce ((a, b) => a + b).

1. **Spark DataFrames**

DataFrame là một Tập dữ liệu 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. API DataFrame có sẵn trong Scala, Java, Python và R. Trong Scala và Java, DataFrame được đại diện bởi Tập dữ liệu của các hàng. Trong API Scala, DataFrame chỉ đơn giản là một bí danh kiểu của Dataset [Row]. Trong khi, trong Java API, người dùng cần sử dụng Dataset <Row> để đại diện cho DataFrame.