BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQHHN

BÁO CÁO KHOA HỌC VỀ DỮ LIỆU LỚN
CHỦ ĐỀ VỀ APACHE SPARK

Nhóm sinh viên thực hiện:

Họ và Tên	Mã sinh viên	Lớp
Nguyễn Trọng Thành (Nhóm Trưởng)	23001934	K68A4
Nguyễn Thành Tâm	23001929	K68A4
Phạm Nhật Quang	23001920	K68A4
Nguyễn Huy Sơn	23001922	K68A4
Nguyễn Minh Thao	23001936	K68A4

Hà Nội ngày 20 tháng 11 năm 2024

LỜI MỞ ĐẦU

Trong thời đại số hóa ngày nay, dữ liệu đã trở thành một tài sản vô giá đối với các tổ chức và doanh nghiệp. Sự phát triển không ngừng của các công nghệ mới như Internet of Things (IoT), mạng xã hội, và các ứng dụng di động đã tạo ra một khối lượng dữ liệu khổng lồ, đòi hỏi các phương pháp xử lý hiệu quả và nhanh chóng để khai thác tiềm năng của nó. Khái niệm "Big Data" xuất hiện như một giải pháp cho việc quản lý và phân tích khối lượng dữ liệu lớn này, giúp doanh nghiệp đưa ra những quyết định chiến lược thông minh hơn, cải thiện dịch vụ khách hàng, và nâng cao hiệu suất hoạt động.

Tuy nhiên, thách thức đặt ra cho các tổ chức là làm thế nào để xử lý và phân tích lượng dữ liệu khổng lồ một cách hiệu quả, đặc biệt khi dữ liệu không chỉ gia tăng về khối lượng mà còn trở nên đa dạng và phức tạp hơn. Việc sử dụng các công cụ truyền thống không còn đủ để đáp ứng yêu cầu xử lý dữ liệu nhanh chóng và linh hoạt trong môi trường hiện đại. Để giải quyết những thách thức này, các hệ thống xử lý dữ liệu phân tán đã được phát triển, trong đó Apache Hadoop từng giữ vai trò tiên phong.

Với sự ra đời của **Apache Spark**, một nền tảng xử lý dữ liệu lớn mạnh mẽ và linh hoạt, việc xử lý và phân tích dữ liệu đã đạt đến một cấp độ mới. Spark nổi bật với tốc độ xử lý nhanh gấp nhiều lần so với Hadoop nhờ khả năng xử lý dữ liệu trong bộ nhớ (in-memory computing) và kiến trúc tối ưu cho việc tính toán phân tán. Ngoài ra, Spark còn tích hợp các thư viện mạnh mẽ cho học máy (MLlib), xử lý dữ liệu thời gian thực (Spark Streaming), truy vấn dữ liệu (Spark SQL), và phân tích đồ thị (GraphX), giúp các nhà phát triển và chuyên gia dữ liệu dễ dàng xây dựng các ứng dụng phân tích phức tạp và đa dạng.

Báo cáo này nhằm cung cấp một cái nhìn toàn diện về Apache Spark, bắt đầu từ tổng quan về Big Data và các phương pháp xử lý, đến giới thiệu chi tiết về kiến trúc và các thành phần của Spark. Bên cạnh đó, báo cáo sẽ hướng dẫn các bước cài đặt Spark trên các nền tảng phổ biến và trình bày các ứng dụng thực tế của nó trong phân tích dữ liệu lớn và học máy. Cuối cùng, một so sánh cụ thể giữa Apache Spark và Apache Hadoop sẽ giúp làm rõ những ưu và nhược điểm của hai hệ thống, từ đó đưa ra cái nhìn khách quan về vị trí của Spark trong thế giới dữ liệu lớn.

Mục Lục

Phần 1. Big Data

1.1 Khái niệm

Big Data là các tập dữ liệu có khối lượng lớn và phức tạp. Độ lớn đến mức các phần mềm xử lý dữ liệu truyền thống không có khả năng thu thập, quản lý và xử lý dữ liệu trong một khoảng thời gian hợp lý.

Big Data không chỉ đề cập đến kích thước của dữ liệu mà còn bao gồm tốc độ, sự đa dạng và tính xác thực của dữ liệu.

Những tập dữ liệu lớn này có thể bao gồm các dữ liệu có cấu trúc, không có cấu trúc và bán cấu trúc, mỗi tập có thể được khai thác để tìm hiểu insights.

1.2 Đặc điểm

Big Data không chỉ đơn thuần là dữ liệu lớn mà còn bao gồm nhiều đặc điểm và tính chất quan trọng giúp hiểu rõ hơn về cách quản lý, phân tích và khai thác dữ liệu. Đặc điểm nổi bật như:

- Dữ liệu được tạo ra và truyền tải với tốc độ rất nhanh từ các nguồn trực tuyến, mạng xã hội, cảm biến và hệ thống giao dịch.
- Khối lượng dữ liệu thường là hàng terabyte (TB), petabyte (PB) hoặc thậm chí exabyte (EB).
- Dữ liệu trong Big Data đến từ nhiều nguồn khác nhau và ở nhiều định dạng khác nhau, bao gồm dữ liệu có cấu trúc (structured), dữ liệu bán cấu trúc (semi-structured) và dữ liệu phi cấu trúc (unstructured).
- Giá trị là mục tiêu cuối cùng của việc khai thác Big Data (Dữ liệu lớn có thể mang lại những hiểu biết và thông tin giá trị giúp các tổ chức và doanh nghiệp đưa ra các quyết định chiến lược, tối ưu hóa hoạt động và tạo ra lợi thế cạnh tranh).
- Lưu lượng dữ liệu có thể thay đổi đáng kể theo thời gian, làm cho việc quản lý và phân tích dữ liệu trở nên phức tạp hơn.

1.3 Ứng dụng thực tế

Big data và phân tích có thể được áp dụng trong nhiều vấn đề kinh doanh và nhiều trường hợp sử dụng khác nhau. Dữ liệu lớn (Big Data) trên thực tế đang được ứng dụng vào rất nhiều

lĩnh vực của nền kinh tế, tạo những chuyển biến ấn tượng, giúp tăng hiệu quả và năng suất của doanh nghiệp.

1.3.1 Ngân hàng

Hình 1. Ứng dụng của BigData trong ngân Hàng.

Trong hệ thống ngân hàng, Big Data đã và đang được ứng dụng hiệu quả thể hiện vai trò quan trọng của mình trong mọi hoạt động của ngân hàng: từ thu tiền mặt đến quản lý tài chính. Ngân hàng ứng dụng Big Data như thế nào:

- Sử dụng các kỹ thuật phân cụm giúp đưa ra quyết định quan trọng. Hệ thống phân tích có thể xác định các địa điểm chi nhánh nơi tập trung nhiều nhu cầu của khách hàng tiềm năng, để đề xuất lập chi nhánh mới.
- Kết hợp nhiều quy tắc được áp dụng trong các lĩnh vực ngân hàng để dự đoán lượng tiền mặt cần thiết sẵn sàng cung ứng ở một chi nhánh tại thời điểm cụ thể hàng năm.
- Khoa học dữ liệu hiện đang là nền tảng của hệ thống ngân hàng kĩ thuật số.
- Machine learning và AI đang được nhiều ngân hàng sử dụng để phát hiện các hoạt động gian lận và báo cáo cho các chuyên viên liên quan.
- Khoa học dữ liệu hỗ trợ xử lý, lưu trữ và phân tích lượng dữ liệu khổng lồ từ các hoạt động hàng ngày và giúp đảm bảo an ninh cho ngân hàng.

1.3.2 Thương mại điện tử

Hình 2. Ứng dụng BigData trong thương mại điện tử.

Thương mại điện tử không chỉ tận hưởng những lợi ích của việc điều hành trực tuyến mà còn phải đối mặt với nhiều thách thức để đạt được các mục tiêu kinh doanh. Lý do là bởi các doanh nghiệp dù là nhỏ hay lớn, khi đã tham gia vào thị trường này đều cần đầu tư mạnh để cải tiến công nghệ. Big Data có thể tạo lợi thế cạnh tranh cho doanh nghiệp bằng cách cung cấp thông tin chuyên sâu và các bản báo cáo phân tích xu hướng tiêu dùng.

Thương mại điện tử ứng dụng Big Data:

- Có thể thu thập dữ liệu và yêu cầu của khách hàng ngay cả trước khi khách thực sự bắt đầu giao dịch.
- Tạo ra một mô hình tiếp thị hiệu suất cao.
- Nhà quản lý trang thương mại điện tử có thể xác định các sản phẩm được xem nhiều nhất và tối ưu thời gian hiển thị của các trang sản phẩm này.
- Đánh giá hành vi của khách hàng và đề xuất các sản phẩm tương tự. Điều này làm tăng khả năng bán hàng, từ đó tạo ra doanh thu cao hơn.
- Nếu bất kỳ sản phẩm nào được thêm vào giỏ hàng nhưng cuối cùng không được khách hàng mua, Big Data có thể tự động gửi code khuyến mại cho khách hàng cụ thể đó.
- Các ứng dụng Big Data còn có thể tạo một báo cáo tùy chỉnh theo các tiêu chí: độ tuổi, giới tính, địa điểm của khách truy cập, v.v…

- Xác định các yêu cầu của khách hàng, những gì họ muốn và tập trung vào việc cung cấp dịch vụ tốt nhất để thực hiện nhu cầu của họ.
- Phân tích hành vi, sự quan tâm của khách hàng và theo xu hướng của họ để tạo ra các sản phẩm hướng đến khách hàng.
- Cung cấp các sản phẩm tốt hơn với chi phí thấp hơn.
- Có thể thu thập nhiều dữ liệu về hành vi khách hàng để thiết kế mô hình tiếp thị tối ưu dành được tùy biến theo đối tượng hoặc nhóm đối tượng, tăng khả năng bán hàng.
- Tìm ra sự tương đồng giữa khách hàng và nhu cầu của họ. Từ đó, việc nhắm mục tiêu các chiến dịch quảng cáo có thể được tiến hành dễ dàng hơn dựa trên những phân tích đã có trước đó.

1.3.3 Y tế

Hình 3. Ứng dụng BigData trong y tế.

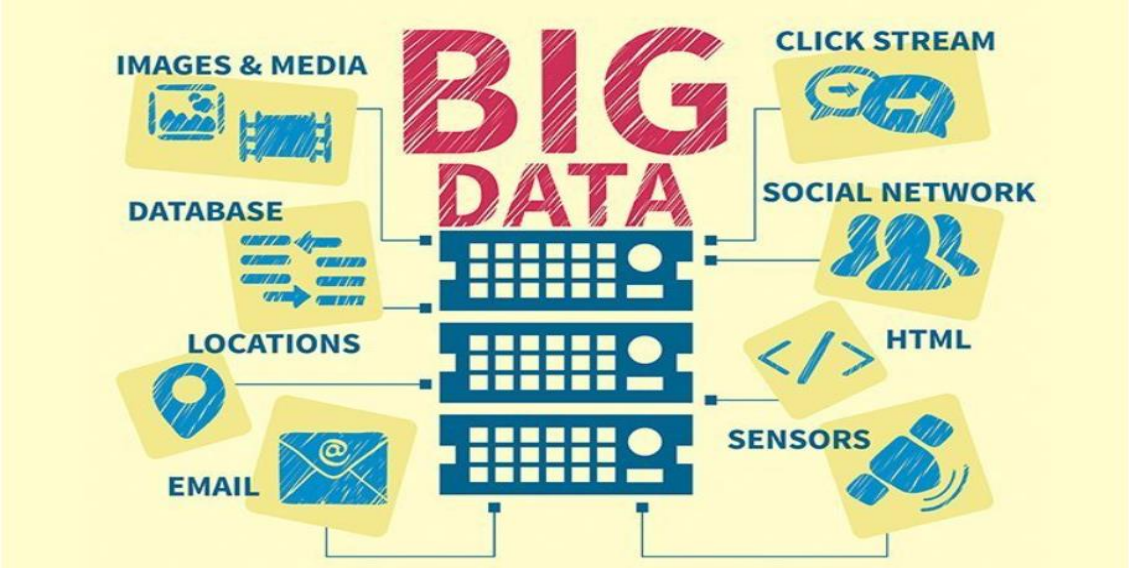
Khoa học dữ liệu đang dần khẳng định vai trò khá quan trọng trong việc cải thiện sức khỏe con người ngày nay. Big Data không chỉ được ứng dụng để xác định phương hướng điều trị mà giúp cải thiện quá trình chăm sóc sức khỏe.

Ngành y tế ứng dụng Big Data:

- Cho phép người quản lý ca dự đoán các bác sĩ cần thiết vào những thời điểm cụ thể.
- Theo dõi tình trạng bệnh nhân bằng để theo dõi hồ sơ sức khỏe điện tử.
- Sử dụng các thiết bị kỹ thuật số có thể đeo, hệ thống Big Data có thể theo dõi bệnh nhân và gửi báo cáo cho các bác sĩ liên quan.

- Big Data có thể đánh giá các triệu chứng và xác định nhiều bệnh ở giai đoạn đầu.
- Có thể lưu giữ các hồ sơ nhạy cảm được bảo mật và lưu trữ lượng dữ liệu khổng lồ một cách hiệu quả.
- Các ứng dụng Big Data cũng có thể báo trước khu vực có nguy cơ bùng phát dịch như: sốt xuất huyết hoặc sốt rét.

1.3.4 Digital Marketing



Hình 4. Ứng dụng BigData trong marketing.

Digital Marketing là chìa khóa để cánh cửa thành công cho bất kỳ doanh nghiệp nào. Giờ đây, không chỉ các công ty lớn có thể điều hành các hoạt động quảng cáo tiếp thị mà cả các doanh nhân nhỏ cũng có thể chạy các chiến dịch quảng cáo thành công trên các nền tảng truyền thông xã hội và quảng bá sản phẩm của họ. Big Data đã tiếp sức cho Digital Marketing phát triển thực sự mạnh mẽ, và nó đã trở thành một phần không thể thiếu của bất kỳ doanh nghiệp nào.

Digital Marketing ứng dụng Big Data:

- Phân tích thị trường, đối thủ cạnh tranh và đánh giá mục tiêu kinh doanh. Điều này giúp cho doanh nghiệp xác định rõ hơn, đâu là cơ hội tốt để tiếp tục tiến hành các kế hoạch kinh doanh tiếp theo.
- Có thể xác định người dùng trên các phương tiện truyền thông xã hội và nhắm mục tiêu cho họ dựa trên nhân khẩu học, giới tính, thu nhập, tuổi tác và sở thích.

- Tạo báo cáo sau mỗi chiến dịch quảng cáo bao gồm hiệu suất, sự tham gia của khán giả và những gì có thể được thực hiện để tạo kết quả tốt hơn.
- Khoa học dữ liệu được sử dụng cho các khách hàng nhắm mục tiêu và nuôi dưỡng chu trình khách hàng.
- Tập trung vào các chủ đề được tìm kiếm cao và tư vấn cho các chủ doanh nghiệp thực hiện chúng trên chiến lược nội dung để xếp hạng trang web doanh nghiệp trên cao hơn trên google (SEO).
- Có thể tạo đối tượng tương tự bằng cách sử dụng cơ sở dữ liệu đối tượng hiện có để nhắm mục tiêu các khách hàng tương tự và kiếm được lợi nhuận.

1.4 Phương pháp, thuật toán sử lý BigData

Trong xử lý Big Data, có nhiều phương pháp và thuật toán khác nhau tùy thuộc vào mục tiêu và dữ liệu cần xử lý. Ở đây chúng tôi xin trình bày những thuật toán phổ biến:

1.4.1 Thuật toán phân lớp Naïve Bayes

1.4.1.1 Mô tả

Bộ phân lớp Naive bayes hay bộ phân lớp Bayes (simple byes classifier) hoạt động như sau:

1. Gọi D là tập dữ liệu huấn luyện, trong đó mỗi phần tử dữ liệu X được biểu diễn bằng một vector chứa n giá trị thuộc tính $A_1, A_2,...,A_n = \{x_1,x_2,...,x_n\}$.
2. Giả sử có m lớp $C_1, C_2,..,C_m$. Cho một phần tử dữ liệu X, bộ phân lớp sẽ gán nhãn cho X là lớp có xác suất hậu nghiệm lớn nhất. Cụ thể, bộ phân lớp Bayes sẽ dự đoán X thuộc vào lớp C_i nếu và chỉ nếu:
   $P(C_i|X) > P(C_j|X)$ $(1<= i, j <=m, i != j)$
   Giá trị này sẽ tính dựa trên định lý Bayes.
3. Để tìm xác suất lớn nhất, ta nhận thấy các giá trị P(X) là giống nhau với mọi lớp nên không cần tính. Do đó ta chỉ cần tìm giá trị lớn nhất của $P(X|C_i) * P(C_i)$. Chú ý rằng $P(C_i)$ được ước lượng bằng $|D_i|/|D|$, trong đó D_i là tập các phần tử dữ liệu thuộc lớp C_i. Nếu xác suất tiền nghiệm $P(C_i)$ cũng không xác định được thì ta coi chúng bằng nhau $P(C_1) = P(C_2) = ... = P(C_m)$, khi đó ta chỉ cần tìm giá trị $P(X|C_i)$ lớn nhất.
4. Khi số lượng các thuộc tính mô tả dữ liệu là lớn thì chi phí tính toàn $P(X|C_i)$ là rất lớn, dó đó có thể giảm độ phức tạp của thuật toán Naive Bayes giả thiết các thuộc tính độc lập nhau.

Khi đó ta có thể tính:

$P(X|C_i) = P(x_1|C_i)...P(x_n|C_i)$

1.4.1.2 Ưu điểm

Giả định độc lập: hoạt động tốt cho nhiều bài toán/miền sữ liệu và ứng dụng.Đơn giản nhưng đủ tốt để giải quyết nhiều bài toán như phân lớp văn bản, lọc spam,..ho phép kết hợp tri thức tiền nghiệm (prior knowledge) và dữ liệu quan sát được (obserwed data).Tốt khi có sự chệnh lệch số lượng giữa các lớp phân loại.Huấn luyện mô hình (ước lượng tham số) dễ và nhanh.

Ví dụ:

Phân các bệnh nhân thành 2 lớp ung thư và không ung thư. Giả sử xác suất để một người bị ung thư là 0.008 tức là P(cancer) = 0.008; và P(nocancer) = 0.992. Xác suất để bệnh nhân ung thư có kết quả xét nghiệm dương tính là 0.98 và xác suất để bệnh nhân không ung thư có kết quả dương tính là 0.03 tức là P(+/cancer) = 0.98, P(+/nocancer) = 0.03. Bây giờ giả sử một bệnh nhân có kết quả xét nghiệm dương tính.

Ta có:

P(+/canncer)P(cancer) = 0.98 * 0.008 = 0.0078

P(+/nocancer)P(nocancer) = 0.03 * 0.992 = 0.0298

Như vậy, P(+/nocancer)P(nocancer) >> P(+/cancer)P(cancer).

Do đó ta xét đoán rằng, bệnh nhân là không ung thư.

1.4.1.3 Công cụ phổ biến

Các công cụ phổ biến của thuật toán phân lớp Naïve Bayes:

- Scikit-Learn (Python)
- Weka (Java)
- Apache Mahout,R, …..

1.4.2 Thuật toán học máy (Machine learning)

1.4.2.1 Mô tả

Quy trình triển khai thuật toán học máy thường bao gồm 6 bước như sau:

1. Thu thập dữ liệu (Gathering data/Data collection)
2. Tiền xử lý dữ liệu (Data preprocessing)

- Trích xuất dữ liệu – Data extraction
- Làm sạch dữ liệu – Data cleaning
- Chuyển đổi dữ liệu – Data transformation
- Chuẩn hóa dữ liệu – Data normalization
- Trích xuất đặc trưng – Feature extraction

3. Phân tích dữ liệu (Data analysis)
4. Xây dựng mô hình máy học (Model building)
5. Huấn luyện mô hình (Model training)
6. Đánh giá mô hình (Model evaluation)

1.4.2.2 Ưu điểm

- Xác định xu hướng dữ liệu dễ dàng
- Khả năng tự động hóa cao xử lý đa dạng dữ liệu

1.4.2.3 Các công cụ phổ biến

Các công cụ phổ biến như:

- Apache Spark
- Amazon SageMaker
- RapidMiner

1.4.3 Phương pháp sử lý luồng dữ liệu

1.4.3.1 Mô tả

Stream Processing là quá trình xử lý dữ liệu gần như ngay lập tức sau khi nó được tạo ra. Điều này đặc biệt hữu ích khi cần xử lý và phân tích dữ liệu theo thời gian thực.

1.4.3.2 Ưu điểm

Stream Processing cung cấp sự linh hoạt trong việc phân tích và xử lý dữ liệu theo thời gian thực.

1.4.3.3 Các công cụ phổ biến

Apache Kafka, Apache Flink, Storm và Google Cloud Dataflow.

Ví dụ:

- Chăm sóc sức khỏe: giám sát liên tục dữ liệu thiết bị.

- An ninh mạng, phát hiện bất thường: xử lý nhật ký web hoặc khủng bố mạng.
- Giao thông vận tải: tối ưu hóa lộ trình và mức tiêu thụ nhiên liệu.

1.4.4 Phương pháp sử lý theo lô

1.4.4.1 Mô tả

Phương pháp sử lí luồng theo lô là việc xử lý một tập dữ liệu lớn mà không cần tương tác người dùng trong quá trình xử lý. Điều này thường được thực hiện với những tác vụ đòi hỏi thời gian xử lý lớn, chẳng hạn như phân tích dữ liệu lớn, xử lý hóa đơn hàng loạt, hoặc thậm chí là xử lý giao dịch ngân hàng hàng loạt.

1.4.4.2 Ưu điểm

Batch Processing phù hợp với các tác vụ đòi hỏi xử lý lượng dữ liệu lớn và không cần kết quả ngay lập tức.

1.4.4.3 Các công cụ phổ biến

Hadoop, Spark, Hive và MapReduce.

1.5 Công cụ sử lý BigData

Trong phần trước, chúng tôi đã trình bày về BigData và tầm quan trọng của nó trong đời sống thực tiễn, sau đây chúng tôi xin trình bày về các công cụ hàng đầu về việc sử lí BigData.

1.5.1 Apache Hadoop

Apache Hadoop là một trong những công cụ được sử dụng phổ biến nhất. Hadoop là một bộ khung mã nguồn mở từ Apache và chạy trên phần cứng. Nó được sử dụng để lưu trữ quá trình và phân tích dữ liệu. Hadoop được viết bằng Java.

Apache Hadoop cho phép xử lý dữ liệu song song khi nó hoạt động trên nhiều máy cùng một lúc. Nó sử dụng cấu trúc cụm. Cụm là một nhóm các hệ thống được kết nối qua mạng LAN.

Nó bao gồm 3 phần:

- Hệ thống tệp phân tán Hadoop (HDFS) – Đây là lớp lưu trữ của Hadoop.
- Map-Reduce – Đây là lớp xử lý dữ liệu của Hadoop.
- YARN – Đây là lớp quản lý tài nguyên của Hadoop.

Mọi công cụ được phát triển đi kèm với một số nhược điểm. Và Hadoop có một số nhược điểm sau đây:

- Hadoop không hỗ trợ xử lý thời gian thực. Nó chỉ hỗ trợ xử lý hàng loạt.
- Hadoop không thể thực hiện các phép tính trong bộ nhớ.

1.5.2 Apache Spark

Apache Spark có thể được coi là sự kế thừa của Hadoop khi nó khắc phục được những nhược điểm của Hadoop. Spark, không giống như Hadoop, hỗ trợ cả thời gian thực cũng như xử lý hàng loạt. Nó là một hệ thống phân cụm mục đích chung.

Nó cũng hỗ trợ tính toán trong bộ nhớ, khiến Apache Spark nhanh hơn 100 lần so với Hadoop. Điều này được thực hiện bằng cách giảm số lượng thao tác đọc/ ghi vào đĩa. Nó cung cấp sự linh hoạt hơn so với Hadoop vì nó hoạt động với các kho dữ liệu khác nhau như HDFS, OpenStack và Apache Cassandra.

Nó cung cấp các API cấp cao trong Java, Python, Scala và R. Spark cũng cung cấp một bộ công cụ cấp cao đáng kể bao gồm Spark SQL để xử lý dữ liệu có cấu trúc, MLlib cho Machine Learning, GraphX để xử lý tập dữ liệu đồ thị và Spark Streaming. Nó cũng bao gồm 80 toán tử cấp cao để thực hiện truy vấn hiệu quả.

1.5.3 Apache Cassandra

Apache Cassandra là hệ cơ sở dữ liệu phân tán, kết hợp những gì tinh tuý nhất của Google Bigtable và Amazon DynamoDB. Ngôn ngữ phát triển Cassandra là Java. Đây là một trong những công cụ dữ liệu lớn tốt nhất có thể chứa tất cả các loại tập dữ liệu cụ thể có cấu trúc, bán cấu trúc và không cấu trúc.

Cassandra được thiết kế có thể chạy trong phần cứng giá rẻ, và cung cấp write throughput khá là cao (latency tầm 0.5ms), trong khi read throughput thì thấp hơn (latency tầm 2.5ms).

1.5.4 MongoDB

MongoDB là một công cụ phân tích dữ liệu nguồn mở, cơ sở dữ liệu NoQuery cung cấp các khả năng đa nền tảng. Đây là công cụ dành cho doanh nghiệp cần dữ liệu nhanh chóng và thời gian thực để đưa ra quyết định.

MongoDB là công cụ hoàn hảo cho những người muốn các giải pháp dựa trên dữ liệu. Nó thân thiện với người dùng vì nó cung cấp cài đặt và bảo trì dễ dàng hơn. MongoDB là công cụ đáng tin cậy và tiết kiệm chi phí.

Nó được viết bằng C, C ++ và JavaScript. Đây là một trong những cơ sở dữ liệu phổ biến nhất cho Big Data vì nó tạo điều kiện thuận lợi cho việc quản lý dữ liệu phi cấu trúc hoặc dữ liệu thay đổi thường xuyên.

MongoDB sử dụng các lược đồ động. Do đó, bạn có thể chuẩn bị dữ liệu nhanh chóng. Điều này cho phép giảm chi phí tổng thể. Nó thực thi trên ngăn xếp phần mềm MEAN, các ứng dụng NET và, nền tảng Java. Nó cũng linh hoạt trong cơ sở hạ tầng đám mây.

1.5.5 RapidMiner

Rapid Miner là một nền tảng phần mềm khoa học dữ liệu cung cấp một môi trường tích hợp để chuẩn bị dữ liệu, học máy, học sâu, khai thác văn bản và phân tích dự đoán. Đây là một trong những hệ thống mã nguồn mở hàng đầu cho khai thác dữ liệu.

Chương trình được viết hoàn toàn bằng ngôn ngữ lập trình Java. Chương trình cung cấp một tùy chọn để thử xung quanh với một số lượng lớn các toán tử tùy ý có thể lồng được chi tiết trong các tệp XML và được thực hiện với sự can thiệp của người dùng đồ họa của người khai thác nhanh.

Những công cụ Big Data kể trên không chỉ giúp bạn lưu trữ số lượng lớn dữ liệu mà còn giúp xử lý dữ liệu được lưu trữ một cách nhanh hơn và cung cấp cho bạn kết quả tốt hơn. Đa số các công cụ Big Data đã có sẵn trên thị trường. Bạn chỉ cần chọn công cụ phù hợp với dự án của bạn.

1.5.6 Apache Storm

Apache Storm là hệ thống tính toán phân tán mã nguồn mở thời gian thực miễn phí. Nếu như Hadoop xử lý dữ liệu hàng loạt (Batch Processing) thì Apache Storm thực hiện xử lý dữ liệu luồng (Unbounded streams of data) một cách đáng tin cậy.

Ưu điểm lớn nhất của Apache Storm là dễ triển khai và có thể tương tác với bất kỳ ngôn ngữ lập trình nào.

Mặt khác, nó đảm bảo việc xử lý từng bộ dữ liệu. Tốc độ xử lý của nó rất nhanh và một tiêu chuẩn có thể quan sát được là tới một triệu tuple được xử lý mỗi giây trên mỗi nút.

Phần 2. Apache Spark

2.1 Giới thiệu và Định nghĩa

Apache Spark là một framework xử lý dữ liệu phân tán mạnh mẽ và linh hoạt. Với kiến trúc phân tán, nó cho phép xử lý các tác vụ dữ liệu lớn và tính toán phân tán trên nhiều node.

Spark được phát triển bắt đầu từ năm 2009 tại AMPLab của Đại học California, Berkeley và sau đó được chuyển giao cho Quỹ phần mềm Apache để quản lý và phát triển tiếp.

Spark được coi là một công cụ xử lý dữ liệu nhanh hơn gấp 10 lần so với các công cụ khác. Nó cung cấp khả năng tính toán phân tán trên các cụm máy tính với hiệu suất cao hơn so với Apache Hadoop và đặc biệt là MapReduce. Spark cũng hỗ trợ nhiều ngôn ngữ lập trình như Scala, Java, Python, giúp cho người dùng dễ dàng phát triển ứng dụng trên nền tảng này.

Triết lý thiết kế của Spark xoay quanh bốn đặc điểm chính:

- Tốc độ
- Dễ sử dụng
- Tính mô-đun
- Khả năng mở rộng

1. Về tính tốc độ (Speed)

Spark nổi bật với khả năng xử lý dữ liệu nhanh chóng nhờ vào công nghệ In-Memory Computing (xử lý dữ liệu ngay trong bộ nhớ). Điều này giúp giảm tối đa độ trễ khi xử lý dữ liệu lớn, đặc biệt trong các ứng dụng phân tích và học máy. Bên cạnh đó, Spark tối ưu hóa khả năng tính toán song song và phân tán, giúp thực hiện các tác vụ một cách nhanh chóng.

2. Về tính dễ sử dụng (Ease of use)

Spark cung cấp **API đơn giản và dễ học** cho nhiều ngôn ngữ lập trình phổ biến như Python, Java, Scala và R. Nhờ vào đó, lập trình viên và nhà khoa học dữ liệu có thể dễ dàng triển khai các giải pháp trên Spark mà không cần quá nhiều kiến thức về hệ thống phân tán. Spark còn tích hợp các công cụ như Spark SQL, Spark Streaming, MLlib để hỗ trợ nhiều loại công việc trong cùng một môi trường.

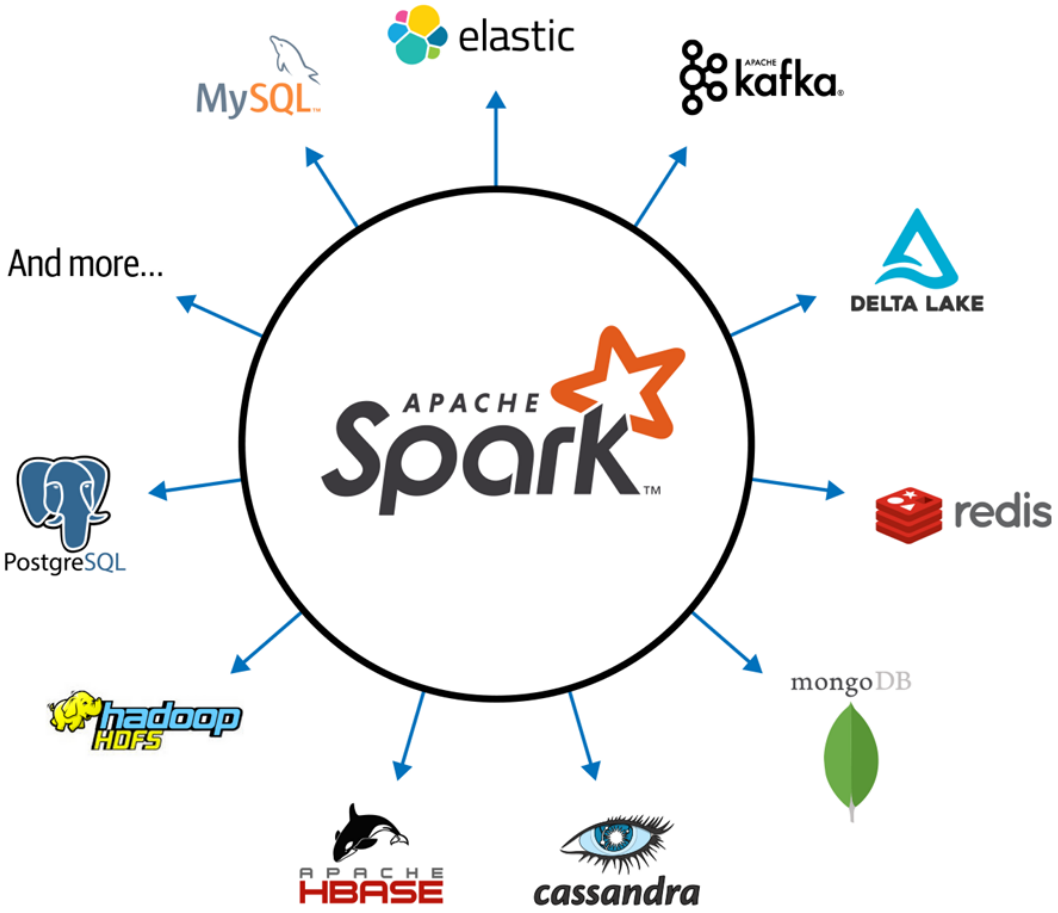
3. Về tính Mô-đun (Modularity)

Spark được xây dựng theo kiến trúc module, giúp người dùng có thể tận dụng từng thành phần riêng biệt như SQL, Streaming, GraphX, và MLlib. Tính mô-đun này cho phép mở rộng và

tích hợp các tính năng dễ dàng khi cần, cũng như tối ưu hóa chi phí và tài nguyên khi triển khai các dịch vụ khác nhau trong hệ sinh thái Spark.

4. Về tính mở rộng (Extensibility)

Spark được thiết kế để mở rộng dễ dàng từ các cụm máy tính nhỏ đến các hệ thống lớn với hàng ngàn máy chủ. Khả năng mở rộng này được thực hiện nhờ vào kiến trúc phân tán của Spark, giúp nó có thể tận dụng tối đa tài nguyên trong các môi trường phân tán, từ các cụm máy tại chỗ đến các nền tảng đám mây như AWS, Azure, và Google Cloud.

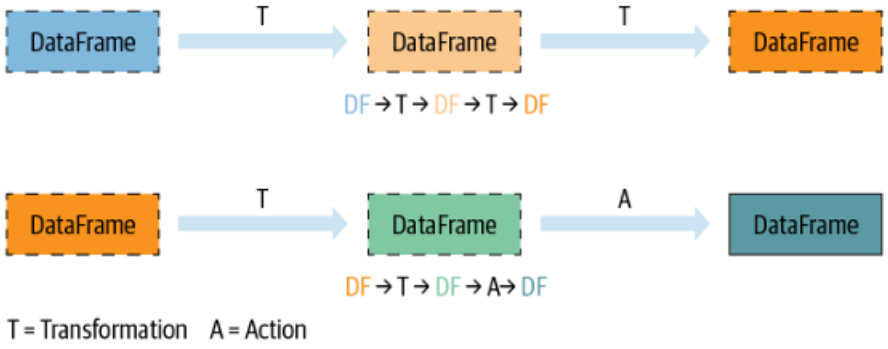


Hình 5. Hệ sinh thái xung quanh Apache Spark.

Các hoạt động Spark trên dữ liệu phân tán có thể được phân thành hai loại: biến đổi và hành động. Các phép biến đổi, như tên cho thấy, chuyển đổi Spark DataFrame thành một DataFrame mới mà không làm thay đổi dữ liệu gốc, mang lại cho nó thuộc tính bất biến. Nói

cách khác, một thao tác như select() hoặc filter() sẽ không thay đổi DataFrame gốc; thay vào đó, nó sẽ trả về kết quả đã chuyển đổi của hoạt động dưới dạng DataFrame mới.

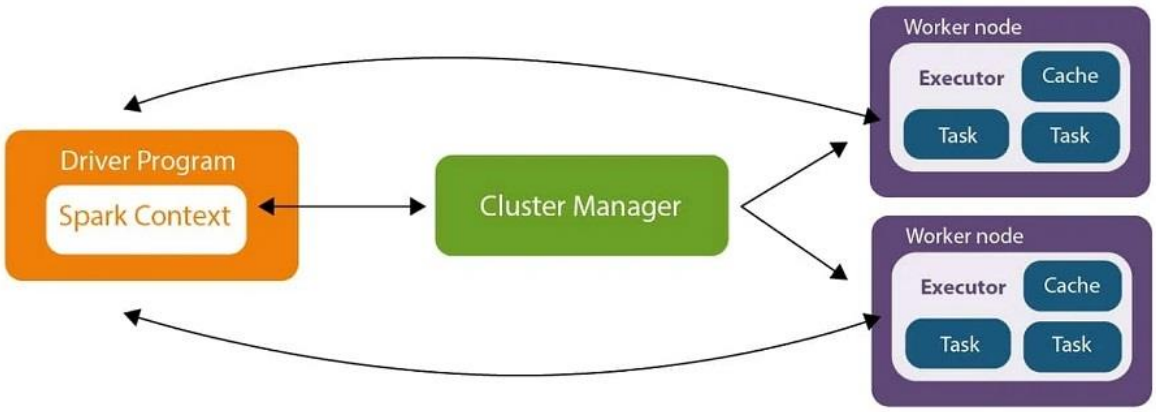
Tất cả các biến đổi được đánh giá một cách lười biếng. Đó là, kết quả của họ không được tính toán một cách chính xác, nhưng chúng được ghi lại hoặc ghi nhớ như một dòng dõi. Một dòng dõi được ghi lại cho phép Spark, sau này trong kế hoạch thực hiện của nó, sắp xếp lại các biến đổi nhất định, làm tổn thương chúng hoặc tối ưu hóa các biến đổi thành các giai đoạn để thực hiện hiệu quả hơn. Đánh giá lười biếng là chiến lược của Spark để trì hoãn việc thực hiện cho đến khi một hành động được gọi hoặc dữ liệu được "chạm" (đọc hoặc ghi vào đĩa). Một hành động kích hoạt đánh giá lười biếng của tất cả các biến đổi được ghi lại. Trong Hình 2-6, tất cả các phép biến đổi T được ghi lại cho đến khi hành động A được gọi. Mỗi phép chuyển đổi T tạo ra một DataFrame mới.



2.2 Kiến trúc của Apache Spark

Kiến trúc của Apache Spark bao gồm các thành phần chính như:

- RDDs
- Driver program
- Spark Context
- Cluster Manager
- Executors



Hình 6. Kiến trúc của Apache Spark.

Các thành phần này hoạt động cùng nhau để quản lý và thực thi quy trình xử lý dữ liệu trên một môi trường phân tán. Điều này giúp Apache Spark đạt được hiệu suất cao, khả năng mở rộng và độ tin cậy trong việc xử lý dữ liệu lớn và tính toán phân tán.

2.2.1 Resilient Distributed Datasets (RDD)

2.2.1.1 Định nghĩa

Resilient Distributed Datasets (RDDs) là một cấu trúc dữ liệu cơ bản và quan trọng trong Apache Spark, cho phép xử lý dữ liệu phân tán trên các cụm máy tính một cách linh hoạt và hiệu quả. Đây là một phần chính của nền tảng Apache Spark, giúp đơn giản hóa việc xử lý dữ liệu lớn bằng cách tự động xử lý phân tán và chịu sự cố.

2.2.1.2 Vai trò

RDD có vai trò to lớn trong Spark:

- Lưu trữ dữ liệu phân tán
- Hỗ trợ khả năng chịu lỗi
- Tính toán song song
- Tích hợp bộ nhớ (in memory) để tính toán nhanh hơn.

2.2.1.3 Về khả năng chịu lỗi và phục hồi dữ liệu của RDD

Nguyên lý phục hồi RDD dữ trên khả năng chịu lỗi (fault tolerance) và dòng dõi (linega). Tức khi một RDD bị lỗi khi đang thực hiện các tranform, … thì spark sẽ sử dụng lenega để tái tạo lại dữ liệu thay vì sao lưu lại, nguyên lý này giúp spark sử lý dữ liệu lớn một cách đáng tin cậy và hiệu quả trong môi trường phân tán.

Nguyên lý phục hồi RDD

- Lineage (Dòng dõi)

Mỗi RDD lưu trữ thông tin về cách nó được tạo ra từ các RDD khác, thông qua các phép biến đổi (transformations) như map, filter, reduce, v.v. Đây là một dòng dõi (lineage) của RDD.

Lineage là một biểu đồ các phép biến đổi, cho phép Spark tái tạo lại một RDD từ các RDD gốc nếu một phần của nó bị mất do lỗi.

Khi một node hoặc một phân vùng của RDD bị mất, Spark không cần phải đọc lại toàn bộ dữ liệu từ nguồn (như HDFS), thay vào đó nó sẽ sử dụng thông tin lineage để tái tạo lại phân vùng bị mất từ các RDD trước đó.

- Phục hồi bằng cách tái tính toán

Khi một phân vùng của RDD bị mất, Spark không sao lưu dữ liệu đó mà chỉ lưu trữ thông tin về cách dữ liệu được tạo ra. Do đó, Spark có thể tái tạo lại dữ liệu từ đầu bằng cách áp dụng lại các phép biến đổi từ lineage.

Ví dụ: Nếu bạn có một RDD rdd được tạo ra từ một file và áp dụng một phép filter để tạo ra filteredRdd, nếu phần của filteredRDD bị mất, Spark sẽ sử dụng thông tin lineage để lấy lại dữ liệu từ RDD gốc và áp dụng lại phép filter.

- Fault Tolerance (Khả năng chịu lỗi)

Spark đảm bảo tính nhất quán và tính chịu lỗi của RDD bằng cách tái tạo lại dữ liệu bị mất mà không làm gián đoạn toàn bộ chương trình. Điều này giúp Spark xử lý được các lỗi trong môi trường phân tán mà không cần sao lưu đầy đủ dữ liệu.

Spark dựa vào Lineage Graph để theo dõi các phép biến đổi được thực hiện trên RDD và tái tạo lại dữ liệu từ các bước biến đổi trước đó.

- Caching và Persisting

Để giảm bớt việc phải tái tính toán lại dữ liệu từ đầu trong các lần sử dụng sau, Spark hỗ trợ caching và persisting các RDD.

Khi một RDD được cache hoặc persist, Spark sẽ lưu trữ dữ liệu đã tính toán vào bộ nhớ hoặc đĩa (tùy chọn). Điều này giúp giảm thời gian tính toán và phục hồi khi một phần của RDD bị mất.

2.2.1.4 Cách khởi tạo

Có ba cách để tạo RDD:

- Song song hóa một tập hợp hiện có trong chương trình trình điều khiển của bạn.
- Tham chiếu một tập dữ liệu trong hệ thống lưu trữ bên ngoài, chẳng hạn như hệ thống tệp được chia sẻ, HDFS, Database, HBase hoặc bất kỳ nguồn dữ liệu nào cung cấp Hadoop InputFormat.
- RDD cũng có thể được tạo ra từ 1 RDD khác.

Các bộ sưu tập song song được tạo bằng cách gọi phương thức JavaSparkContext's parallelize trên một Collectionchương trình trình điều khiển hiện có của bạn. 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 bộ sưu tập song song chứa các số từ 1 đến 5:

```
SparkConf conf = new SparkConf().setAppName("application").setMaster("local");
JavaSparkContext sc = new JavaSparkContext(conf);
List<Integer> data = Arrays.asList(1, 2, 3, 4, 5);
```

RDD tệp văn bản có thể được tạo bằng phương thức SparkContext's textFile. Phương thức này lấy URI cho tệp (hoặc đường dẫn cục bộ trên máy hoặc URI hdfs://, s3a://, v.v.) và đọc nó như một tập hợp các dòng. Sau đây là một ví dụ về lệnh gọi:

```
// RDD was created by Spark Context
JavaRDD<String> newData = sc.textFile( path: "C:\\Users\\Trong Thanh\\Documents\\BIG DATA\\spark_4\\src\\main\\java\\org\\example");
System.out.println(newData.getClass().getName());
newData.foreach(line -> System.out.println(line));
```

Spark có thể tạo các tập dữ liệu phân tán từ bất kỳ nguồn lưu trữ nào được Hadoop hỗ trợ, bao gồm hệ thống tệp cục bộ, HDFS, Cassandra, HBase, Amazon S3 , v.v. Spark hỗ trợ các tệp văn bản, SequenceFiles và bất kỳ Hadoop InputFormat nào khác .

RDD được tạo ra từ một RDD khác, ví dụ như:

```
List<Integer> data = Arrays.asList(1, 2, 3, 4, 5);
JavaRDD<Integer> distData = sc.parallelize(data);

JavaRDD<Integer> newData = distData.map(x -> x * 2);
```

2.2.2 Driver program

2.2.2.1 Định nghĩa

Driver Program là thành phần chính trong ứng dụng Spark. Nó chịu trách nhiệm khởi tạo SparkContext, quản lý tiến trình thực thi của ứng dụng, phân phối công việc đến các Executor (thực thi) và theo dõi tiến độ của các tác vụ.

2.2.2.2 Vai trò

- Driver Program quản lý tiến trình thực thi của ứng dụng, phân phối công việc đến các Executor
- Driver Program chịu trách nghiệm khởi tạo Spark Context:

```
SparkConf conf = new SparkConf().setAppName("application").setMaster("local");
JavaSparkContext sc = new JavaSparkContext(conf);
```

2.2.3 Spark Context

2.2.3.1 Định nghĩa

Spark Context là điểm vào (entry point) chính để tương tác với các chức năng của Spark. Nó chịu trách nhiệm kết nối ứng dụng với cụm Spark, quản lý tài nguyên, và cung cấp các API để thực hiện các phép toán trên dữ liệu.

Thành phần chính:

- Configuration (cấu hình): Spark Context sử dụng Lớp SparkConf để thiết lập các cấu hình cho ứng dụng, ví dụ như setName(), bộ nhớ, số lõi,..
- Scheduler (bộ lập lịch): phân phối tài nguyên đến các executors

2.2.3.2 Vai trò

Vai trò của Spark Context:

- Tạo kết nối đến với Cluster để yêu cầu quản lý tài nguyên người dùng.
- Khởi tạo RDD
- Phân phối tác vụ đến các Executor
- …
- Environment (môi trường): gồm những thông tin về môi trường, thông tin về phiên bản của Spark, thư viện được sử dụng, …

2.2.4 Cluster Manager

2.2.4.1 Định nghĩa

Cluster manager (trình quản lý cụm) là một thành phần trong hệ thống cluster, chịu trách nhiệm quản lý và điều phối các tài nguyên trong cụm máy tính (cluster). Nhiệm vụ chính của cluster manager bao gồm phân bổ tài nguyên, quản lý các node, giám sát các tác vụ và điều phối việc thực hiện công việc giữa các node trong cụm.

2.2.4.2 Vai trò

Vai trò của Cluster manager:

- Phân tán tải công việc: Cluster giúp chia nhỏ khối lượng công việc lớn và phân phối chúng cho nhiều máy tính trong cụm, từ đó tối ưu hóa tài nguyên và tăng hiệu suất.
- Tăng cường khả năng mở rộng: Khi nhu cầu xử lý tăng cao, ta có thể dễ dàng thêm nhiều máy tính (node) vào cluster mà không làm gián đoạn hệ thống.
- Tăng độ tin cậy và sẵn sàng cao: Nếu một node trong cluster gặp sự cố, các node khác có thể tiếp quản nhiệm vụ mà không ảnh hưởng đến hoạt động chung. Điều này giúp hệ thống luôn sẵn sàng và giảm thiểu thời gian chết (downtime).
- Tăng tốc độ xử lý: Cluster cho phép xử lý song song nhiều tác vụ cùng lúc, tăng tốc độ xử lý so với việc chạy trên một máy tính đơn lẻ.
- Quản lý dữ liệu phân tán: Cluster thường được sử dụng để quản lý các hệ thống cơ sở dữ liệu phân tán, giúp xử lý dữ liệu lớn (Big Data) hoặc các tác vụ yêu cầu xử lý lượng dữ liệu khổng lồ.

2.2.5 Executors

2.2.5.1 Định nghĩa

Executors là các tiến trình (processes) chạy trên các node của cluster.

Thành phần của Executors bao gồm task Execution để sử lí các phép tính cụ thể, block manager để quản lý dữ liệu.

2.2.5.2 Vai trò

Vai trò của Executors là:

- Thực thi các task

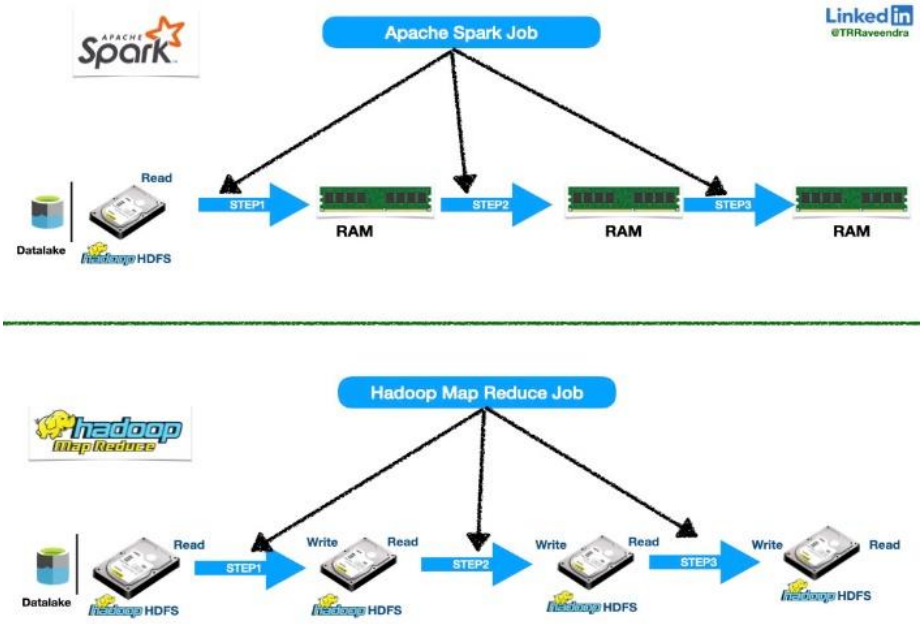
- Quản lý dữ liệu bộ nhớ
- Tương tác với Driver

2.3 Thành phần và thư viện của Spark

2.3.1 Spark Core

2.3.1.1 Định nghĩa

Apache Spark Core là thành phần trung tâm của Apache Spark, một nền tảng xử lý dữ liệu phân tán tốc độ cao. Spark Core cung cấp các chức năng cơ bản cho việc xử lý dữ liệu phân tán, đảm bảo khả năng chịu lỗi và làm nền tảng cho các thành phần khác của Spark như Spark SQL, Spark Streaming, MLlib và GraphX.



Hình 7. Minh họa cách hoạt động của Apache Spark và Apache Hadoop.

2.3.1.2 Tính năng

Một trong những tính năng chính của Spark Core là RDD (Resilient Distributed Dataset). RDD là cấu trúc dữ liệu cơ bản trong Spark, đại diện cho một tập dữ liệu phân tán trên nhiều node (máy chủ) trong cluster. RDD hỗ trợ hai loại thao tác chính: Transformation và Action. Các thao tác Transformation như map() và filter() cho phép tạo ra RDD mới từ một RDD hiện có, trong khi các thao tác Action như collect() và count() tính toán kết quả dựa trên RDD và trả về kết quả cho người dùng.

Một đặc điểm quan trọng khác của Spark Core là tính toán trễ (Lazily Evaluated). Các phép biến đổi trên RDD không được thực hiện ngay lập tức mà được lưu lại dưới dạng DAG (Directed Acyclic Graph). Chỉ khi người dùng thực hiện các thao tác Action, Spark mới bắt đầu tiến hành tính toán dựa trên DAG này, giúp tối ưu hóa quá trình xử lý dữ liệu.

Về quản lý cluster, Spark Core có khả năng tích hợp với các hệ thống quản lý như Hadoop YARN, Mesos, hoặc có thể sử dụng cluster riêng của Spark. Điều này cho phép Spark dễ dàng mở rộng quy mô để xử lý lượng dữ liệu lớn một cách hiệu quả. Bên cạnh đó, Spark Core cũng có bộ lập lịch công việc tích hợp, giúp chia nhỏ công việc thành các tác vụ (task) và phân phối chúng đến các node trong cluster một cách tối ưu.

Spark Core hỗ trợ nhiều ngôn ngữ lập trình khác nhau như Scala, Java, Python và R mang lại sự linh hoạt cho các lập trình viên.

2.3.2 Spark SQL

2.3.2.1 Định nghĩa

Spark SQL là một mô-đun Spark để xử lý dữ liệu có cấu trúc. Không giống như Spark RDD API cơ bản, các giao diện do Spark SQL cung cấp cung cấp cho Spark nhiều thông tin hơn về cấu trúc của cả dữ liệu và phép tính đang được thực hiện. Về mặt nội bộ, Spark SQL sử dụng thông tin bổ sung này để thực hiện các tối ưu hóa bổ sung. Có một số cách để tương tác với Spark SQL bao gồm SQL và Dataset API. Khi tính toán kết quả, cùng một công cụ thực thi được sử dụng, bất kể bạn đang sử dụng API/ngôn ngữ nào để thể hiện phép tính. Sự hợp nhất này có nghĩa là các nhà phát triển có thể dễ dàng chuyển đổi qua lại giữa các API khác nhau dựa trên cách cung cấp cách tự nhiên nhất để thể hiện một phép biến đổi nhất định.

Một công dụng của Spark SQL là thực thi các truy vấn SQL. Spark SQL cũng có thể được sử dụng để đọc dữ liệu từ một cài đặt Hive hiện có. Để biết thêm về cách cấu hình tính năng này, vui lòng tham khảo phần Hive Tables . Khi chạy SQL từ bên trong một ngôn ngữ lập trình khác, kết quả sẽ được trả về dưới dạng Dataset/DataFrame . Bạn cũng có thể tương tác với giao diện SQL bằng dòng lệnh hoặc qua JDBC/ODBC.

Note:

- Hive: là một công cụ cơ sở hạ tầng kho dữ liệu để xử lý dữ liệu có cấu trúc trong Hadoop. Nó nằm trên đỉnh Hadoop để tóm tắt Dữ liệu lớn và giúp truy vấn và phân tích dễ dàng.
- JDBC/ODBC: là các API để truy cập các hệ quản trị cơ sở dữ liệu (DBMS).

2.3.2.2 Datasets và DataFrame

Bộ dữ liệu và khung dữ liệu (Datasets và DataFrame):

Dataset là một tập hợp dữ liệu phân tán. Dataset là một giao diện mới được thêm vào Spark 1.6 cung cấp các lợi ích của RDD (kiểu dữ liệu mạnh, khả năng sử dụng các hàm lambda mạnh mẽ) với các lợi ích của công cụ thực thi được tối ưu hóa của Spark SQL. Dataset có thể được xây dựng từ các đối tượng JVM và sau đó được thao tác bằng các phép biến đổi hàm (map, flatMap, filter, v.v.). API Dataset có sẵn trong Scala và Java . Python không hỗ trợ API Dataset. Nhưng do bản chất động của Python, nhiều lợi ích của API Dataset đã có sẵn (tức là bạn có thể truy cập trường của một hàng theo tên một cách tự nhiên row.columnName). Trường hợp của R cũng tương tự.

DataFrame là một Dataset 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 có nhiều tối ưu hóa hơn. DataFrame có thể được xây dựng từ nhiều nguồn khác nhau 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 biểu diễn bằng một Dataset của Rows. Trong API Scala , DataFrame chỉ đơn giản là một bí danh kiểu của Dataset[Row]. Trong khi đó, trong API Java , người dùng cần sử dụng Dataset<Row>để biểu diễn DataFrame.

Tóm gọn lại, Dataset là một Là tập hợp các dữ liệu phân tán, là api được thêm vào Spark để cung cấp các lợi ích của rdd (kiểu dữ liệu mạng cùng với lưu trữ các hàm lamda mạnh mẽ) với các lợi ích công cụ thực thi được tối ưu hóa của Spark SQL.

2.3.2.3 SparkSession

Điểm vào của tất cả các chức năng trong Spark là lớp SparkSession. Để tạo một lớp cơ bản SparkSession, chỉ cần sử dụng SparkSession.builder():

```
val sparkSparkSession = SparkSession.builder().appName("Spark Demo").config("spark.master", "local").getOrCreate()
```

Khởi tạo DataFrames

Với SparkSession, các ứng dụng có thể tạo DataFrame từ dữ liệu hiện có RDD , từ bảng Hive hoặc từ các nguồn dữ liệu Spark .

Ví dụ, đoạn mã sau đây tạo DataFrame dựa trên nội dung của tệp JSON:

```
val sparkSparkSession = SparkSession.builder().appName("Spark Demo").config("spark.master", "local").getOrCreate()

val dataFrame = sparkSparkSession.read.option("multiline","true").json("C:\\Users\\Trong Thanh\\Documents\\TestData\\people.json")

dataFrame.select("people.name").show()
```

```
+-----------+
|       name|
+-----------+
|Trọng Thành|
+-----------+
```

DataFrames cung cấp ngôn ngữ chuyên biệt cho việc xử lý dữ liệu có cấu trúc trong Scala , Java , Python và R.

Như đã đề cập ở trên, trong Spark 2.0, DataFrames chỉ là Dataset của Rows trong Scala và Java API. Các hoạt động này cũng được gọi là "untyped transformations" trái ngược với "typed transformations" đi kèm với các Dataset Scala/Java được typed mạnh.

Một số ví dụ cơ bản về xử lý dữ liệu có cấu trúc bằng cách sử dụng Bộ dữ liệu:

```
// Select everybody, but increment the age by 1
df.select($"name", $"age" + 1).show()
// +-------+----------+
// |   name|(age + 1)|
// +-------+----------+
// |Michael|      null|
// |   Andy|        31|
// | Justin|        20|
// +-------+----------+

// Select people older than 21
df.filter($"age" > 21).show()
// +---+----+
// |age|name|
// +---+----+
// | 30|Andy|
// +---+----+

// Count people by age
df.groupBy("age").count().show()
// +----+-----+
// | age|count|
// +----+-----+
// |  19|    1|
// |null|    1|
// |  30|    1|
// +----+-----+
```

Hàm sql trên SparkSession cho phép các ứng dụng chạy các truy vấn SQL theo chương trình và trả về kết quả dưới dạng DataFrame.

```scala
// Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")

val sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
// +----+-------+
// | age|   name|
// +----+-------+
// |null|Michael|
// |  30|   Andy|
// |  19| Justin|
// +----+-------+
```

2.3.2.4 *Tạo bộ dữ liệu*

Bộ dữ liệu tương tự như RDD, tuy nhiên, thay vì sử dụng Java serialization hoặc Kryo, chúng sử dụng Encoder chuyên dụng để serialize các đối tượng để xử lý hoặc truyền qua mạng. Trong khi cả bộ mã hóa và serialization chuẩn đều có trách nhiệm biến một đối tượng thành byte, thì bộ mã hóa là mã được tạo động và sử dụng định dạng cho phép Spark thực hiện nhiều thao tác như lọc, sắp xếp và băm mà không cần deserialize các byte trở lại thành một đối tượng.

```scala
case class Person(name: String, age: Long)

// Encoders are created for case classes
val caseClassDS = Seq(Person("Andy", 32)).toDS()
caseClassDS.show()
// +----+---+
// |name|age|
// +----+---+
// |Andy| 32|
// +----+---+

// Encoders for most common types are automatically provided by importing spark.implicits._
val primitiveDS = Seq(1, 2, 3).toDS()
primitiveDS.map(_ + 1).collect() // Returns: Array(2, 3, 4)

// DataFrames can be converted to a Dataset by providing a class. Mapping will be done by name
val path = "examples/src/main/resources/people.json"
val peopleDS = spark.read.json(path).as[Person]
peopleDS.show()
// +----+-------+
// | age|   name|
// +----+-------+
// |null|Michael|
// |  30|   Andy|
// |  19| Justin|
// +----+-------+
```

2.3.2.5 Tương tác với RDD

Spark SQL hỗ trợ hai phương pháp khác nhau để chuyển đổi RDD hiện có thành Dataset. Phương pháp đầu tiên sử dụng phản xạ để suy ra lược đồ của RDD chứa các loại đối tượng cụ thể. Phương pháp dựa trên phản xạ này dẫn đến mã ngắn gọn hơn và hoạt động tốt khi bạn đã biết lược đồ khi viết ứng dụng Spark của mình.

Phương pháp thứ hai để tạo Dataset là thông qua giao diện lập trình cho phép bạn xây dựng lược đồ và sau đó áp dụng nó vào RDD hiện có. Mặc dù phương pháp này chi tiết hơn, nhưng nó cho phép bạn xây dựng Dataset khi các cột và kiểu của chúng không được biết cho đến khi chạy.

2.3.2.6 Nguồn dữ liệu

Spark SQL Công cụ này hỗ trợ nhiều nguồn dữ liệu khác nhau, bao gồm JDBC, ODBC, JSON, HDFS, Hive, ORC và Parquet. Các kho dữ liệu phổ biến khác như Amazon Redshift, Amazon S3, Couchbase, Cassandra, MongoDB, Salesforce.com, Elasticsearch và nhiều kho dữ liệu thuộc hệ sinh thái Spark Packages.

Ví dụ:

Với Json

Spark SQL có thể tự động suy ra lược đồ của một tập dữ liệu JSON và tải nó dưới dạng Dataset[Row]. Chuyển đổi này có thể được thực hiện bằng cách sử dụng SparkSession.read.json() trên tệp Dataset[String], hoặc tệp JSON.

Lưu ý rằng tệp được cung cấp dưới dạng *tệp json* không phải là tệp JSON thông thường. Mỗi dòng phải chứa một đối tượng JSON hợp lệ, độc lập và riêng biệt. Để biết thêm thông tin, vui lòng xem định dạng văn bản Dòng JSON, còn được gọi là JSON phân cách bằng dòng mới .

Đối với tệp JSON nhiều dòng thông thường, hãy đặt multiLine tùy chọn thành true.

```scala
case class Person(name: String, age: Long)

// Encoders are created for case classes
val caseClassDS = Seq(Person("Andy", 32)).toDS()
caseClassDS.show()
// +----+---+
// |name|age|
// +----+---+
// |Andy| 32|
// +----+---+

// Encoders for most common types are automatically provided by importing spark.implicits._
val primitiveDS = Seq(1, 2, 3).toDS()
primitiveDS.map(_ + 1).collect() // Returns: Array(2, 3, 4)

// DataFrames can be converted to a Dataset by providing a class. Mapping will be done by name
val path = "examples/src/main/resources/people.json"
val peopleDS = spark.read.json(path).as[Person]
peopleDS.show()
// +----+-------+
// | age|   name|
// +----+-------+
// |null|Michael|
// |  30|   Andy|
// |  19| Justin|
// +----+-------+
```

2.3.3 Spark Streamming

2.3.3.1 Định nghĩa

Spark Streaming là phần mở rộng của API Spark cốt lõi cho phép xử lý luồng dữ liệu trực tiếp có khả năng mở rộng, thông lượng cao, chịu lỗi. Dữ liệu có thể được thu thập từ nhiều nguồn như Kafka, Kinesis hoặc socket TCP và có thể được xử lý bằng các thuật toán phức tạp được thể hiện bằng các hàm cấp cao như map, reduce, join và . Cuối cùng, dữ liệu đã xử lý có thể được đẩy ra hệ thống tệp, cơ sở dữ liệu và bảng điều khiển trực tiếp. Trên thực tế, bạn có thể áp dụng các thuật toán học máy và xử lý đồ thi window của Spark trên các luồng dữ liệu.

Hình 8. Spark Streaming và các nguồn dữ liệu.

2.3.3.2 Cách hoạt động

Về mặt nội bộ, nó hoạt động như sau. Spark Streaming nhận luồng dữ liệu đầu vào trực tiếp và chia dữ liệu thành từng đợt, sau đó được xử lý bởi công cụ Spark để tạo luồng kết quả cuối cùng theo từng đợt.

Hình 9. Cách hoạt động của Spark Streaming.

Spark Streaming cung cấp một sự trừu tượng hóa cấp cao được gọi là *luồng rời rạc* hoặc *DStream* , biểu diễn một luồng dữ liệu liên tục. DStream có thể được tạo từ các luồng dữ liệu đầu vào từ các nguồn như Kafka và Kinesis hoặc bằng cách áp dụng các hoạt động cấp cao trên các DStream khác. Về mặt nội bộ, DStream được biểu diễn dưới dạng một chuỗi RDD .

2.3.3.3 Streaming Context

Để khởi tạo chương trình Spark Streaming, cần phải tạo đối tượng StreamingContext , đây là điểm vào chính của mọi chức năng Spark Streaming.

```scala
import org.apache.spark._
import org.apache.spark.streaming._

val conf = new SparkConf().setAppName(appName).setMaster(master)
val ssc = new StreamingContext(conf, Seconds(1))
```

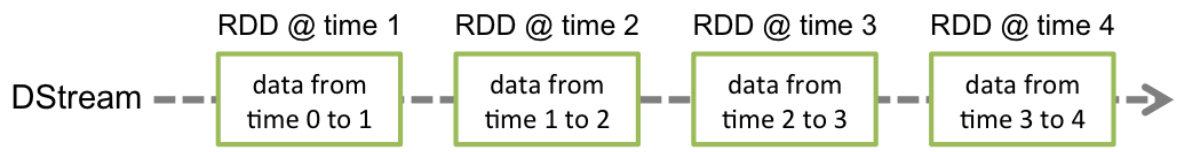
Tham appNameso là tên cho ứng dụng của bạn hiển thị trên UI cụm. masterlà URL cụm Spark, Mesos, Kubernetes hoặc YARN hoặc chuỗi "local[*]" đặc biệt để chạy ở chế độ cục bộ. Trong thực tế, khi chạy trên cụm, bạn sẽ không muốn mã hóa cứng mastertrong chương trình mà muốn khởi chạy ứng dụng bằngspark-submit và nhận ứng dụng tại đó. Tuy nhiên, đối với thử nghiệm cục bộ và thử nghiệm đơn vị, bạn có thể truyền "local[*]" để chạy Spark Streaming trong quá trình (phát hiện số lõi trong hệ thống cục bộ). Lưu ý rằng điều này tạo ra một SparkContext (điểm bắt đầu của tất cả các chức năng Spark) có thể được truy cập dưới dạng ssc.sparkContext.

Một StreamingContextđối tượng cũng có thể được tạo ra từ một SparkContext đối tượng hiện có.

```scala
import org.apache.spark.streaming._

val sc = ...                // existing SparkContext
val ssc = new StreamingContext(sc, Seconds(1))
```

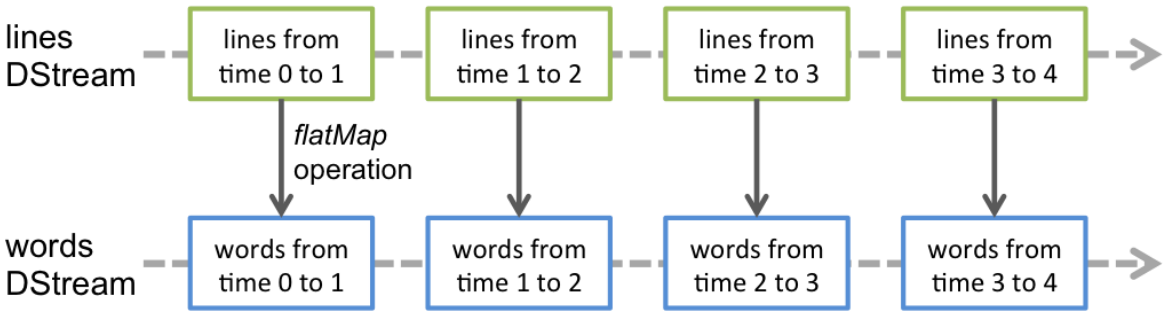
2.3.3.4 DStream

Discreted Stream hay DStream là sự trừu tượng cơ bản do Spark Streaming cung cấp. Nó biểu diễn một luồng dữ liệu liên tục, hoặc là luồng dữ liệu đầu vào nhận được từ nguồn hoặc là luồng dữ liệu đã xử lý được tạo ra bằng cách chuyển đổi luồng đầu vào. Về mặt nội bộ, DStream được biểu diễn bằng một chuỗi liên tục các RDD, là sự trừu tượng của Spark đối với một tập dữ liệu phân tán, không thay đổi (xem Spark Programming Guide để biết thêm chi tiết). Mỗi RDD trong DStream chứa dữ liệu từ một khoảng thời gian nhất định, như thể hiện trong hình sau.



Hình 10. Luồng Streamming

Bất kỳ thao tác nào được áp dụng trên DStream đều được chuyển thành các thao tác trên RDD cơ bản. Ví dụ, trong ví dụ trước về việc chuyển đổi luồng dòng thành từ, flatMap thao tác được áp dụng trên mỗi RDD trong lines DStream để tạo RDD của words DStream. Điều này được thể hiện trong hình sau.



Hình 11. Luồng Streamming

2.3.4 Spark Mlib

2.3.4.1 Định nghĩa

MLlib là thư viện học máy (ML) của Spark. Mục tiêu của nó là làm cho việc học máy thực tế có thể mở rộng và dễ dàng. Ở cấp độ cao, nó cung cấp các công cụ như:

- Thuật toán ML: các thuật toán học tập phổ biến như phân loại, hồi quy, phân cụm và lọc cộng tác
- Đặc điểm hóa: trích xuất đặc điểm, chuyển đổi, giảm chiều và lựa chọn
- Đường ống: công cụ để xây dựng, đánh giá và điều chỉnh Đường ống ML
- Sự bền bỉ: lưu và tải các thuật toán, mô hình và Đường ống
- Tiện ích: đại số tuyến tính, thống kê, xử lý dữ liệu, v.v.

2.3.4.2 Các thuật toán học máy

Trong Spark MLlib, có nhiều thuật toán học máy phục vụ cho các tác vụ khác nhau như hồi quy, phân loại, phân cụm, và giảm chiều dữ liệu. Dưới đây là các thuật toán học máy phổ biến trong MLlib:

Học có giám sát (Supervised Learning):

- Hồi quy tuyến tính
- Hồi quy logistic
- Cây quyết định
- Rừng ngẫu nhiên
- …

Học không giám sát (Unsupervised Learning):

- K-mean
- Gaussian Mixture Models
- …

Xử lý dữ liệu văn bản:

- Word2Vec
- TF-IDF

Ngoài ra còn rất nhiều các thuật toán khác.

2.3.5 Spark GraphX

2.3.5.1 Định nghĩa

GraphX là một thành phần mới trong Spark dành cho đồ thị và tính toán song song đồ thị. Ở cấp độ cao, GraphX mở rộng Spark RDD bằng cách giới thiệu một dạng trừu tượng đồ thi mới : một đồ thị đa hướng có các thuộc tính được gắn vào mỗi đỉnh và cạnh. Để hỗ trợ tính toán đồ thị, GraphX đưa ra một tập hợp các toán tử cơ bản (ví dụ: subgraph , joinVertices và aggregateMessages) cũng như một biến thể được tối ưu hóa của Pregel API. Ngoài ra, GraphX bao gồm một bộ sưu tập ngày càng tăng các thuật toán đồ thị và trình xây dựng để đơn giản hóa các tác vụ phân tích đồ thị.

2.3.5.2 Biểu đồ Thuộc tính

Đồ thi thuộc tính là một đồ thị đa hướng có các đối tượng do người dùng định nghĩa được gắn vào mỗi đỉnh và cạnh. Một đồ thị đa hướng là một đồ thị có hướng có nhiều cạnh song song có khả năng chia sẻ cùng một đỉnh nguồn và đích. Khả năng hỗ trợ các cạnh song song giúp đơn giản hóa các tình huống mô hình hóa trong đó có thể có nhiều mối quan hệ (ví dụ: đồng nghiệp và bạn bè) giữa các đỉnh giống nhau. Mỗi đỉnh được khóa bằng một định danh dài 64 bit *duy*

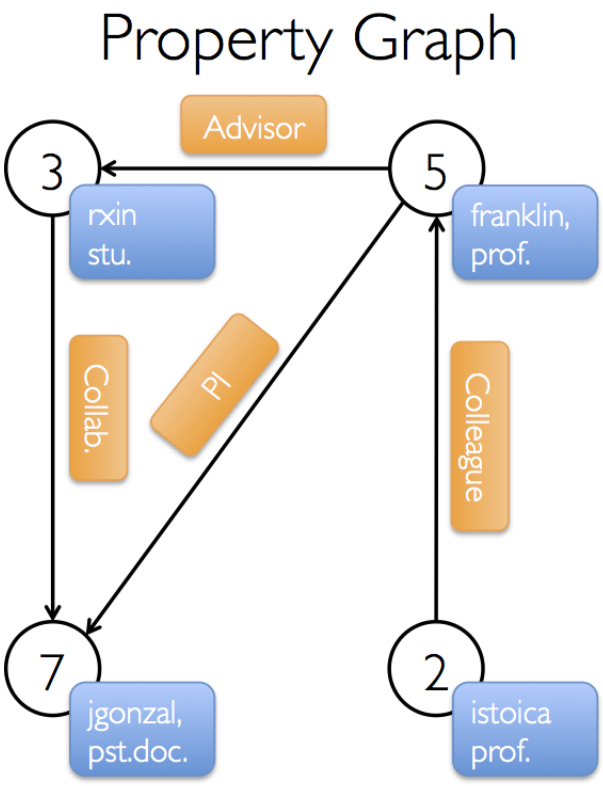
*nhất*VertexId . GraphX không áp đặt bất kỳ ràng buộc sắp xếp nào lên các định danh đỉnh. Tương tự như vậy, các cạnh có các định danh đỉnh nguồn và đích tương ứng.

Đồ thị thuộc tính được tham số hóa qua các kiểu đỉnh (VD) và cạnh (ED). Đây là các kiểu đối tượng được liên kết với từng đỉnh và cạnh tương ứng.

GraphX tối ưu hóa việc biểu diễn các kiểu đỉnh và cạnh khi chúng là các kiểu dữ liệu nguyên thủy (ví dụ: int, double, v.v.) giúp giảm dung lượng bộ nhớ bằng cách lưu trữ chúng trong các mảng chuyên dụng.

Ví dụ về biểu đồ thuộc tính

Giả sử chúng ta muốn xây dựng một đồ thị thuộc tính bao gồm nhiều cộng tác viên khác nhau trên dự án GraphX. Thuộc tính đỉnh có thể chứa tên người dùng và nghề nghiệp. Chúng ta có thể chú thích các cạnh bằng một chuỗi mô tả mối quan hệ giữa các cộng tác viên:



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Hình 11. Đồ thị.

2.3.5.3 Toán tử đồ thị

Cũng giống như RDD có các phép toán cơ bản như map, filter, và reduceByKey, đồ thị thuộc tính cũng có một tập hợp các toán tử cơ bản lấy các hàm do người dùng định nghĩa và tạo

ra các đồ thị mới với các thuộc tính và cấu trúc đã được chuyển đổi. Các toán tử cốt lõi có các triển khai được tối ưu hóa được định nghĩa trong Graphvà các toán tử tiện lợi được biểu thị dưới dạng các hợp phần của các toán tử cốt lõi được định nghĩa trong GraphOps. Tuy nhiên, nhờ Scala ngầm định, các toán tử trong GraphOpstự động có sẵn dưới dạng các thành viên của Graph. Ví dụ, chúng ta có thể tính toán bậc trong của mỗi đỉnh (được định nghĩa trong GraphOps) theo công thức sau:

```
val graph: Graph[(String, String), String]
// Use the implicit GraphOps.inDegrees operator
val inDegrees: VertexRDD[Int] = graph.inDegrees
```

2.3.5.4 Toán tử cấu trúc

Hiện tại GraphX chỉ hỗ trợ một tập hợp đơn giản các toán tử cấu trúc thường dùng và chúng tôi dự kiến sẽ bổ sung thêm trong tương lai.

Toán reverse tử trả về một đồ thị mới với tất cả các hướng cạnh bị đảo ngược. Điều này có thể hữu ích khi, ví dụ, cố gắng tính toán PageRank nghịch đảo. Vì phép toán đảo ngược không sửa đổi các thuộc tính đỉnh hoặc cạnh hoặc thay đổi số lượng cạnh, nên nó có thể được triển khai hiệu quả mà không cần di chuyển hoặc sao chép dữ liệu.

Toán subgraph tử lấy các vị từ đỉnh và cạnh và trả về đồ thị chỉ chứa các đỉnh thỏa mãn vị từ đỉnh (đánh giá là đúng) và các cạnh thỏa mãn vị từ cạnh *và kết nối các đỉnh thỏa mãn vị từ đỉnh* . subgraph Toán tử có thể được sử dụng trong một số tình huống để giới hạn đồ thị ở các đỉnh và cạnh quan tâm hoặc loại bỏ các liên kết bị hỏng.

Toán mask tử xây dựng một đồ thị con bằng cách trả về một đồ thị chứa các đỉnh và cạnh cũng có trong đồ thị đầu vào. Điều này có thể được sử dụng kết hợp với toán subgraphtử để hạn chế một đồ thị dựa trên các thuộc tính trong một đồ thị liên quan khác. Ví dụ, chúng ta có thể chạy các thành phần được kết nối bằng cách sử dụng đồ thị có các đỉnh bị thiếu và sau đó hạn chế câu trả lời cho đồ thị con hợp lệ.

Toán groupEdges tử hợp nhất các cạnh song song (tức là các cạnh trùng lặp giữa các cặp đỉnh) trong đa đồ thị. Trong nhiều ứng dụng số, các cạnh song song có thể được *thêm vào* (trọng số của chúng được kết hợp) thành một cạnh duy nhất, do đó làm giảm kích thước của đồ thị.

2.3.5.5 API Pregel

Đồ thị vốn là cấu trúc dữ liệu đệ quy vì các thuộc tính của đỉnh phụ thuộc vào các thuộc tính của các đỉnh lân cận, mà các đỉnh lân cận này lại phụ thuộc vào các thuộc tính của *các* đỉnh

lân cận. Do đó, nhiều thuật toán đồ thị quan trọng lặp đi lặp lại tính toán lại các thuộc tính của từng đỉnh cho đến khi đạt đến điều kiện điểm cố định. Một loạt các phép trừu tượng song song đồ thị đã được đề xuất để thể hiện các thuật toán lặp đi lặp lại này. GraphX trình bày một biến thể của API Pregel.

2.4 Nguồn dữ liệu

Apache Spark có thể kết nối đến các nguồn dữ liệu khác nhau như:

- Hệ thống tệp JSON
- Cơ sở dữ liệu NoSQL thông qua Cassandra, MongoDB
- Đến các cơ sở dữ liệu thông qua các dirver ODBC (Object Database Connectivity) như JDBC của Java, …
- Hệ thống tệp phân tán của Hadoop là HDFS (Hadoop Distributed File System)
- Hệ thống cloud như Microsoft Azure, Google Cloud Storage, …

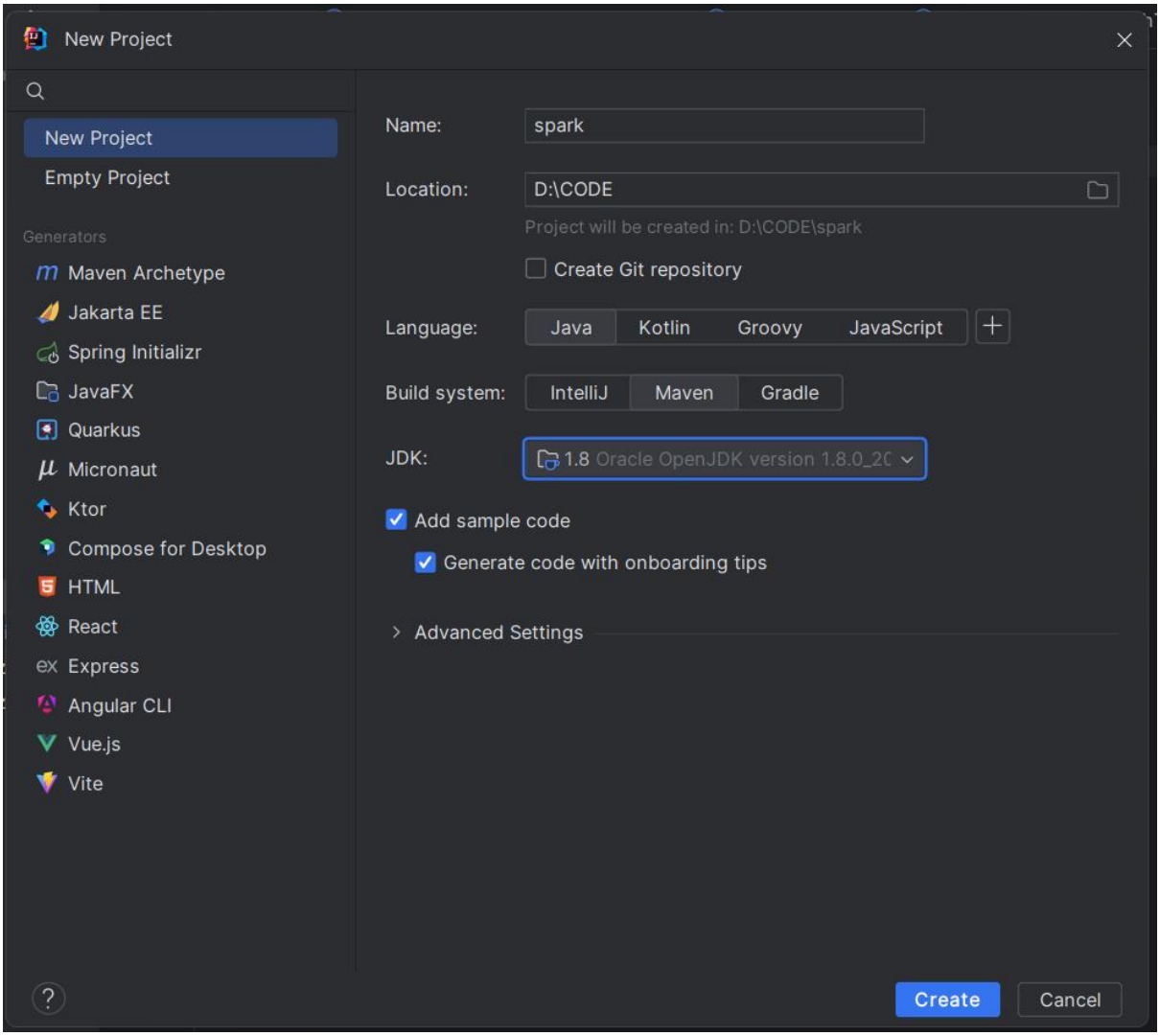
2.5 Cài đặt Spark

Có 2 cách để cài đặt Apache Spark sử dụng chế độ Local và Cluster, ở đây chúng tôi xin trình bày cách cài đặt, cấu hình Apache Spark bằng 2 cách

- Trên môi trường phát triển tích hợp (IDE) cụ thể là trên Intelij IDEA ở Local mode
- Trên trực tiếp máy tính Window ở local mode

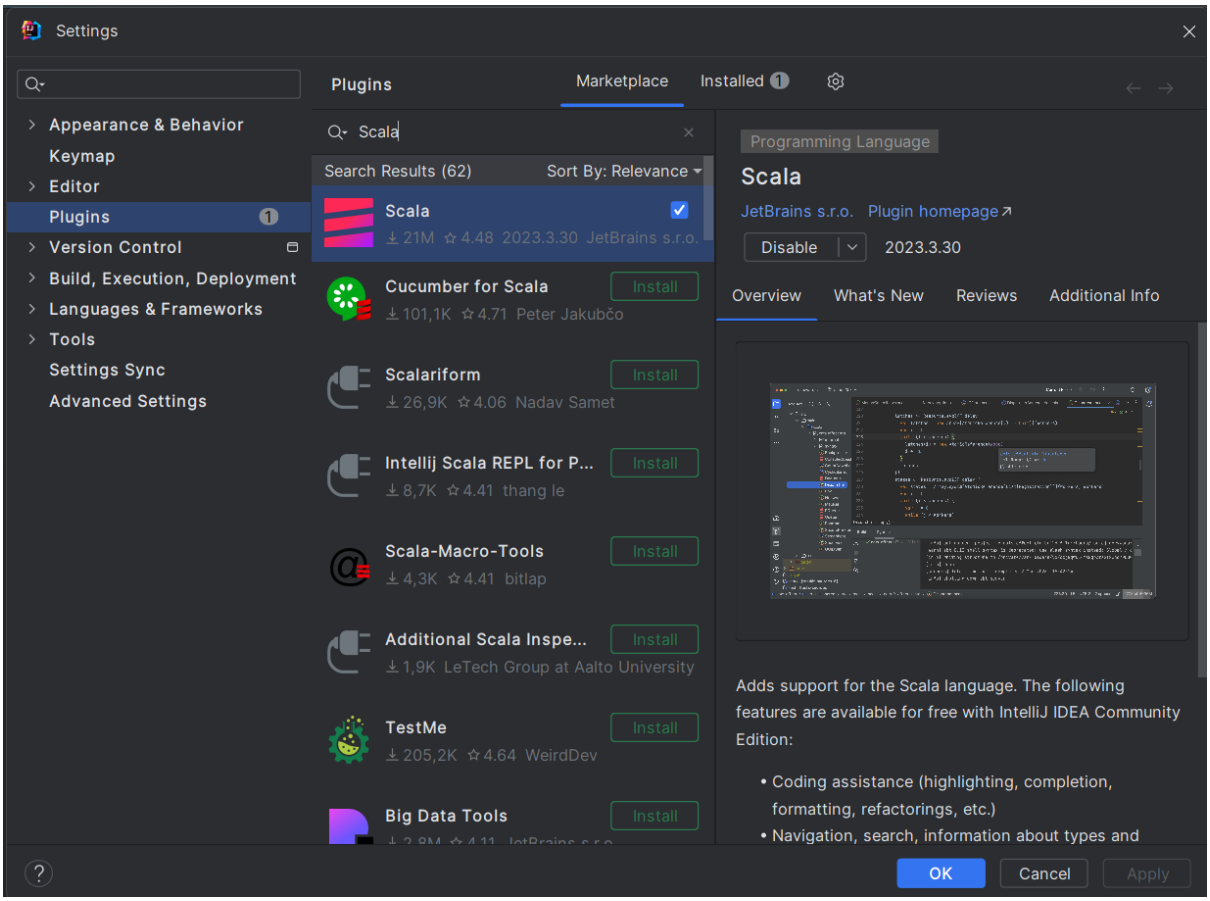
2.5.1 Cài đặt trên Intelij IDEA

1. Tạo project mới với maven



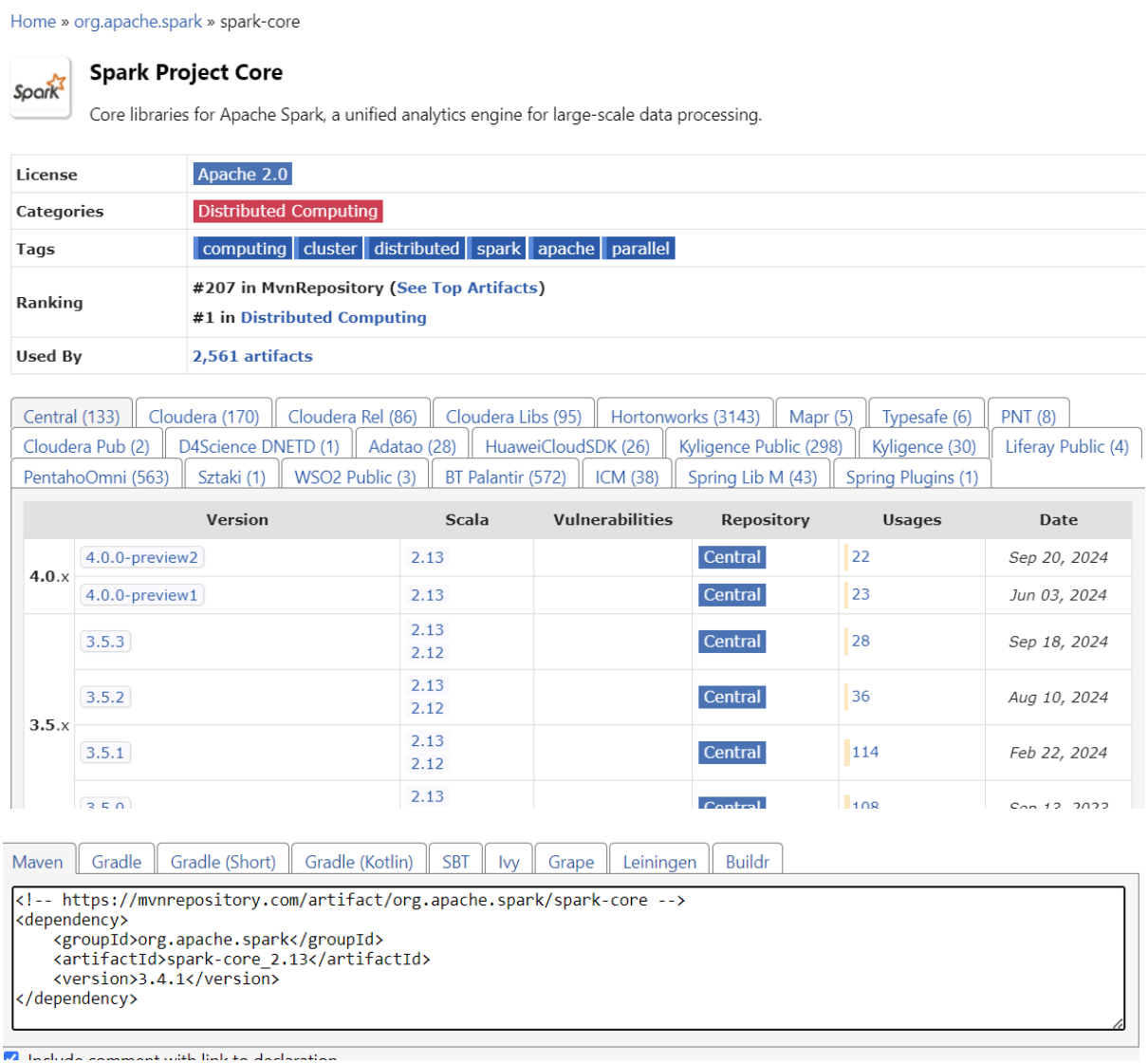
ở đây chúng tôi sử dụng JDK 8 (Java Deverloment Kit), ngoài ra có thể sử dụng JDK 11, JDK 17, một vài phiên bản khác sẽ không thể sử dụng để tạo trương trình với Spark

2. Vào phần cài đặt của IntelliJ -> Chọn Plugins -> Download Scala.



3. Cài đặt Spark

Vào trang web https://mvnrepository.com/ và tìm kiếm thư viện Apache Spark là Spark Project Core.



Tùy chọn một thư viện thích hợp để cài đặt, copy đoạn code xml để vào trong file pom.xml trong project.

Làm tương tự với Spark Sql và thư viện Scala Library.

Home » org.scala-lang » scala-library

Scala Library

Standard library for the Scala Programming Language

License	Apache 2.0
Categories	JVM Languages
Tags	jvm language library scala
Ranking	#6 in MvnRepository (See Top Artifacts) #2 in JVM Languages
Used By	37,618 artifacts

This artifact was moved to:

org.scala-lang » scala3-library

Central (212) | Redhat GA (19) | Redhat EA (3) | D4Science DNETD (2) | ICM (3) | JCenter (1)

Version	Vulnerabilities	Repository	Usages	Date
2.13.15		Central	1,111	Sep 20, 2024
2.13.15-M1		Central	5	Aug 16, 2024
2.13.14		Central	3,481	Apr 30, 2024
2.13.13		Central	2,500	Feb 21, 2024

Search for groups, artifacts, categories — Search

Home » org.apache.spark » spark-sql

Spark Project SQL

Spark SQL is Apache Spark's module for working with structured data based on DataFrames.

License	Apache 2.0
Categories	SQL Libraries
Tags	database sql query spark apache client
Ranking	#223 in MvnRepository (See Top Artifacts) #1 in SQL Libraries
Used By	2,388 artifacts

Central (128) | Cloudera (147) | Cloudera Rel (80) | Cloudera Libs (130) | Hortonworks (4793) | Mapr (5) | Typesafe (6) | WSO2 Releases (3) | PNT (9) | Cloudera Pub (1) | D4Science DNETD (1) | Adatao (19) | HuaweiCloudSDK (24) | Kyligence Public (300) | Kyligence (26) | PentahoOmni (345) | BT Palantir (572) | ICM (16) | Spring Lib M (15) | Spring Lib Release (33) | Spring Plugins (1)

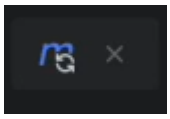
	Version	Scala	Vulnerabilities	Repository	Usages	Date
4.0.x	4.0.0-preview2	2.13		Central	10	Sep 20, 2024
	4.0.0-preview1	2.13		Central	11	Jun 03, 2024
3.5.x	3.5.3	2.13 2.12		Central	19	Sep 18, 2024
	3.5.2	2.13 2.12		Central	26	Aug 10, 2024
	3.5.1	2.13 2.12		Central	115	Feb 22, 2024
	3.5.0	2.13		Central	121	Sep 13, 2023

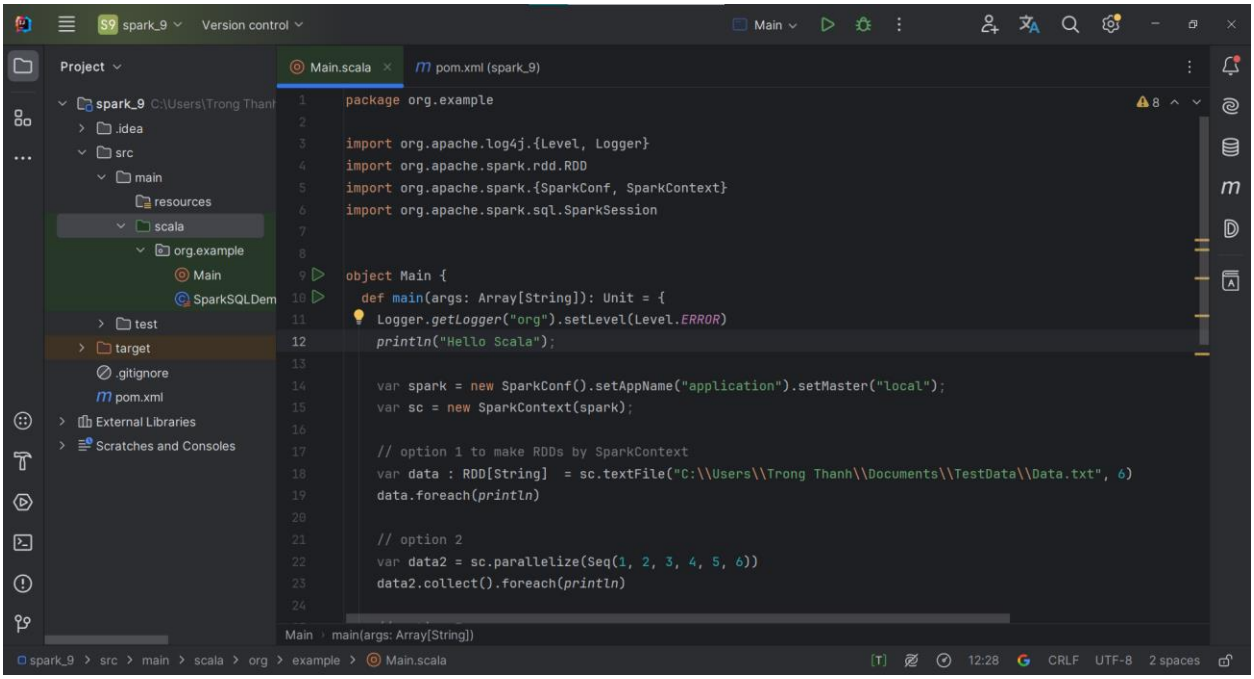
Trong file pom.xml ta sẽ có đoạn cấu hình như sau:

```xml
<properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <spark.version>3.4.1</spark.version>
    <scala.version>2.12.18</scala.version>
</properties>
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.spark/spark-core -->
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.12</artifactId>
        <version>${spark.version}</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.spark/spark-sql -->
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-sql_2.12</artifactId>
        <version>${spark.version}</version>
        <scope>provided</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.scala-lang/scala-library -->
    <dependency>
        <groupId>org.scala-lang</groupId>
        <artifactId>scala-library</artifactId>
        <version>${scala.version}</version>
    </dependency>

</dependencies>
```

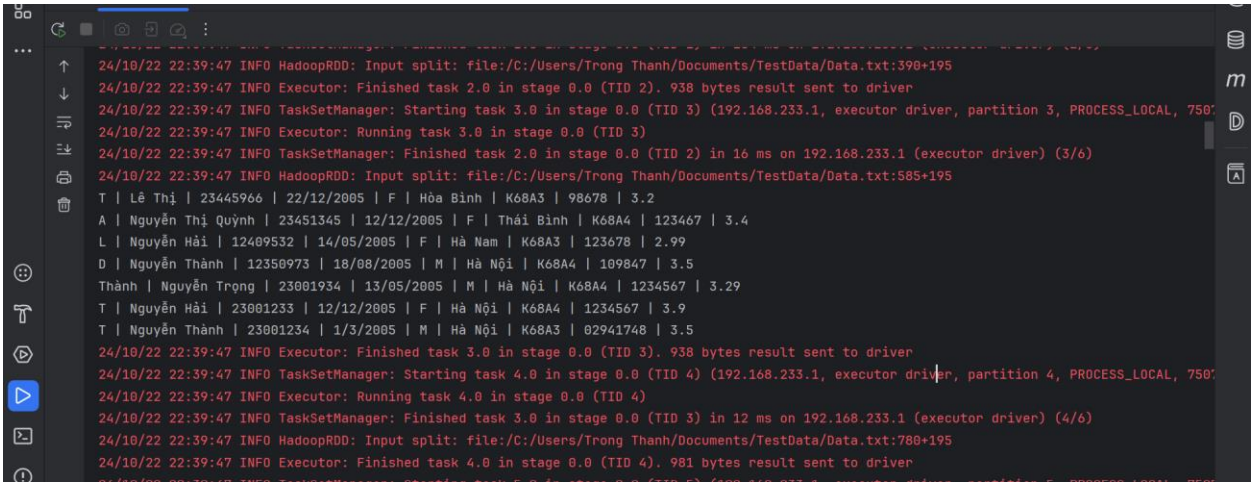
Để trương trình với maven có thể chạy được thì ta thêm đoạn code sau đây để xác định được file nguồn:

```xml
<build>
    <plugins>
        <plugin>

            <groupId>org.codehaus.mojo</groupId>
            <artifactId>build-helper-maven-plugin</artifactId>
            <version>3.6.0</version>
            <executions>
                <execution>
                    <id>add-test-source</id>
                    <phase>generate-test-sources</phase>
                    <goals>
                        <goal>add-test-source</goal>
                    </goals>
                    <configuration>
                        <sources>
                            <!-->source file for project<-->
                            <source>src/main/scala</source>
                        </sources>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

4. Sau đó click vào biểu tượng  trong file cấu hình pom để IDE cài đặt các thư viện đã nêu ở trên
5. Trương trình đầu tiên



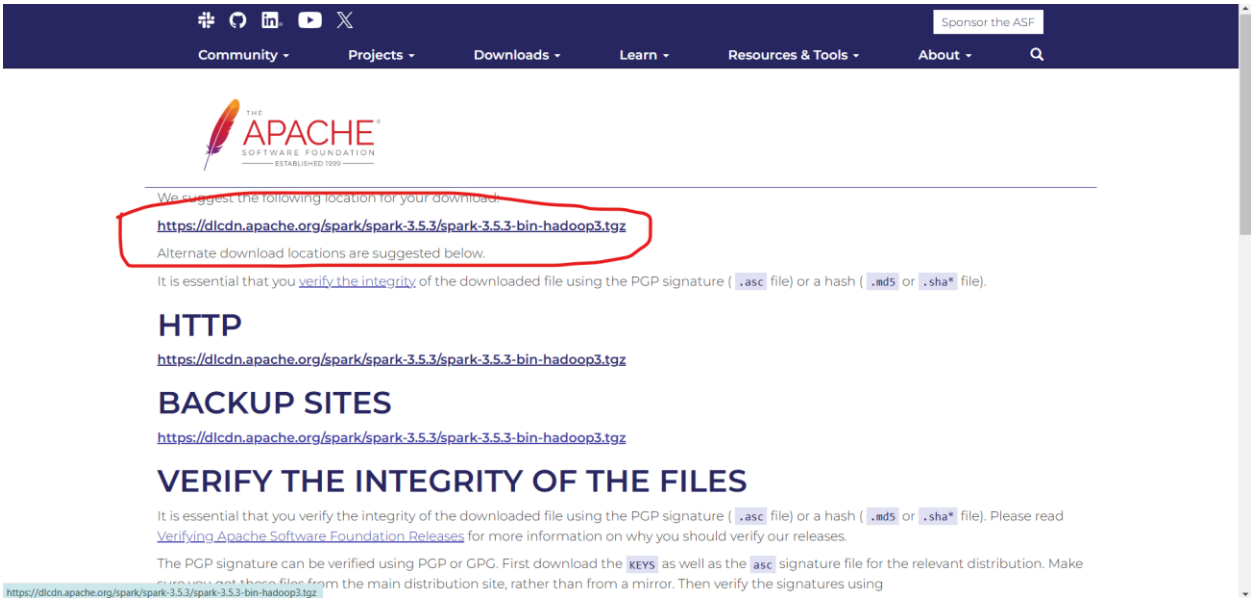
Kết quả:



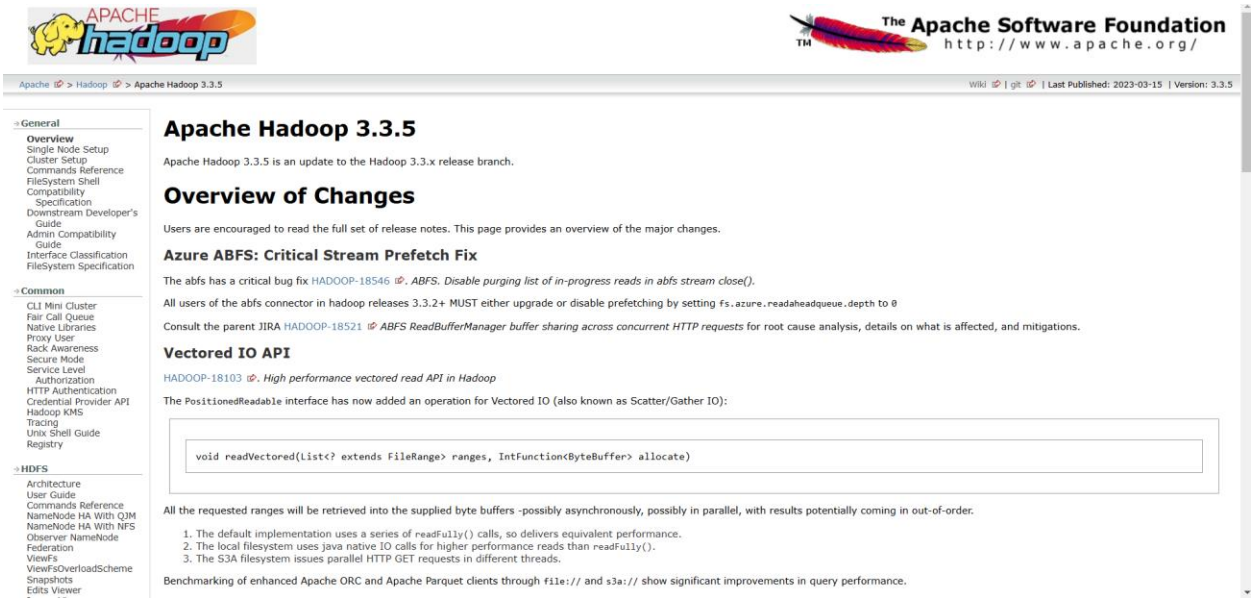
2.5.2 Cài đặt trên trực tiếp Hệ điều hành Window

1. Cài đặt JDK 8/11/17 trên máy
2. Cài đặt Spark

    Ở đây chúng tôi sử dụng phiên bản spark-3.5.3



3. Nếu người dùng muốn sử dụng PySpark hoặc Rspark thì có thể cài thêm Python hoặc R
4. Cài đặt Winutil (Hadoop) phiên bản 3.3.5



5. Thiết lập biến môi trường cho cho các công cụ trên
6. Khởi chạy Spark-shell

Ở đây ta có thể chạy chương trình bằng Cmd hoặc Jupyter Notebook.

Tóm lại, mỗi cách cài có những ưu, nhược điểm khác nhau, tùy thuộc vào mục đích sử dụng của mỗi người.

2.6 Ứng dụng của Spark

Spark có thể được sử dụng trong nhiều kịch bản khác nhau, từ xử lý dữ liệu lớn đến học máy. Các ứng dụng chính của Spark bao gồm:

- Phân tích dữ liệu lớn
- Machine Learnning
- Sử lý thời gian thực
- Sử lý đồ thị
- Và các lĩnh vực khác trong đời sống

2.6.1 Phần tích dữ liệu lớn

Phân tích dữ liệu lớn với Apache Spark hỗ trợ truy vấn SQL, xử lý dữ liệu cấu trúc, phi cấu trúc và tối ưu hóa truy vấn, ứng dụng rộng rãi trong tài chính, marketing, và khoa học dữ liệu.

- Spark SQL Hỗ trợ DataFrames và Dataset API, giúp phân tích tài chính và hành vi khách hàng trên dữ liệu lớn.
- Phân tích log, JSON, và dữ liệu mạng xã hội để tìm lỗi, xu hướng, hoặc cảm xúc.
- Catalyst Optimizer cải thiện hiệu suất cho ETL và phân tích dự đoán.

2.6.2 *Học máy*

Apache Spark hỗ trợ học máy qua thư viện MLlib với khả năng xử lý phân tán dữ liệu lớn.

- Thuật toán phổ biến:
    - Logistic Regression, Random Forest (phân loại email, chẩn đoán bệnh).
    - Linear Regression, Decision Trees (dự đoán giá nhà, doanh thu).
    - K-Means, GMM (phân đoạn khách hàng, phân tích thị trường).
- Ứng dụng: Huấn luyện mô hình trên dữ liệu lớn, phân tích thời gian thực kết hợp Spark Streaming.

2.6.3 *Xử lý thời gian thực*

Apache Spark, với Spark Streaming và Structured Streaming, hỗ trợ xử lý dữ liệu theo luồng từ các nguồn như cảm biến IoT, giao dịch tài chính, và mạng xã hội.

- Ứng dụng:
    - Về iot Giám sát môi trường, phân tích lỗi thiết bị công nghiệp.
    - Về mạng xã hội: Phân tích cảm xúc, theo dõi xu hướng.
    - Tài chính: Xử lý giao dịch chứng khoán, phát hiện gian lận.
    - Thương mại điện tử: Phân tích hành vi khách hàng, cập nhật hệ thống khuyến nghị.
    - Tích hợp: Kết hợp Kafka, HDFS, NoSQL để thu thập log và cảnh báo thời gian thực.
    - Học máy: Phát hiện gian lận, tối ưu hóa trải nghiệm khách hàng theo thời gian thực.

2.6.4 *Ứng dụng khác*

1, Phân tích dữ liệu (Data Analytics)

Spark hỗ trợ các hoạt động phân tích dữ liệu lớn, chẳng hạn như việc phân tích log, phân tích dữ liệu người dùng, và xử lý ETL (Extract, Transform, Load). Tích hợp với SQL thông qua Spark SQL giúp xử lý truy vấn SQL nhanh chóng trên các bộ dữ liệu lớn.

2, Xử lý đồ thị (Graph Processing)

Spark có GraphX, một API dành riêng cho việc xử lý đồ thị và tính toán đồ thị phân tán. Điều này rất hữu ích trong các ứng dụng như phân tích mạng xã hội, phân tích đồ thị đường đi ngắn nhất, hoặc hệ thống gợi ý.

3, Hệ thống khuyến nghị (Recommendation Systems)

Với khả năng xử lý dữ liệu lớn và thư viện học máy mạnh mẽ, Spark là một lựa chọn tuyệt vời cho việc xây dựng các hệ thống gợi ý (recommendation systems), ví dụ như đề xuất sản phẩm, phim ảnh dựa trên sở thích của người dùng.

Phần 3. So sánh với Apache Hadoop

3.1 Điểm giống nhau

Cả Hadoop và Spark đều là các hệ thống phân tán hỗ trợ xử lý dữ liệu lớn với khả năng phục hồi cao khi xảy ra sự cố.

- **Xử lý dữ liệu lớn phân tán**

Hadoop chia nhỏ các tập dữ liệu thành các phân vùng, sau đó lưu trữ và xử lý chúng trên một mạng lưới phân tán gồm nhiều máy chủ. Tương tự, Spark phân tích và xử lý dữ liệu lớn trên các nút phân tán để cung cấp thông tin chi tiết. Cả hai hệ thống đều có thể được tích hợp với các phần mềm khác để tối ưu hóa hiệu năng.

- **Khả năng chịu lỗi cao**

Hadoop lưu trữ nhiều bản sao của dữ liệu trên các nút khác nhau. Khi một nút bị lỗi, hệ thống sẽ truy xuất dữ liệu từ các bản sao còn lại để tiếp tục xử lý.

Spark sử dụng RDD (Tập dữ liệu phân tán linh hoạt) để ghi nhớ cách tái tạo dữ liệu. Điều này giúp Spark khôi phục thông tin nhanh chóng ngay cả khi kho lưu trữ gặp sự cố.

3.2 Điểm khác nhau

3.2.1 Kiến trúc

Apache Spark sử dụng kiến trúc in-memory để lưu trữ dữ liệu trong RAM, giúp xử lý nhanh. Tính chịu lỗi được đảm bảo nhờ DAG và RDD. Spark hỗ trợ nhiều kiểu xử lý như batch, streaming, interactive và machine learning.

Apache Hadoop dựa trên kiến trúc MapReduce, xử lý dữ liệu qua ghi/đọc đĩa, khiến tốc độ chậm hơn. Tính chịu lỗi được đảm bảo qua sao lưu dữ liệu trên HDFS và phù hợp cho xử lý dữ liệu theo lô.

3.2.2 Hiệu năng

Apache Spark xử lý dữ liệu nhanh nhờ lưu trữ và tính toán trực tiếp trên RAM, đặc biệt hiệu quả với các bài toán phức tạp hoặc lặp lại. Spark tối ưu hóa luồng công việc qua DAG, hỗ trợ xử lý song song cao trên nhiều lõi CPU.

Apache Hadoop xử lý dựa trên đĩa cứng, phù hợp với bài toán xử lý theo lô nhưng tốc độ chậm hơn trong các trường hợp cần tính toán phức tạp hoặc nhiều vòng lặp.

3.2.3 Cách dùng

Apache Spark có API dễ sử dụng cho Python, Scala, Java và R, hỗ trợ cả dữ liệu batch và streaming. Spark tương thích với hệ thống Hadoop hiện có và dễ dàng tích hợp các bài toán phức tạp như machine learning.

Apache Hadoop đòi hỏi mã phức tạp hơn khi lập trình với MapReduce, phù hợp cho xử lý dữ liệu theo lô. Hệ sinh thái Hadoop có các công cụ hỗ trợ như Hive, Pig giúp đơn giản hóa quy trình.

3.2.4 Chi phí

Apache Spark yêu cầu nhiều RAM hơn, dẫn đến chi phí phần cứng cao hơn nhưng tiết kiệm chi phí vận hành nhờ tốc độ xử lý nhanh. Spark hỗ trợ mở rộng tốt, giảm thời gian và tài nguyên khi mở rộng.

Ngược lại, Apache Hadoop yêu cầu phần cứng thấp hơn nhưng thời gian xử lý chậm có thể dẫn đến tăng chi phí vận hành. Hadoop thích hợp với lưu trữ dữ liệu lớn nhưng chi phí quản lý tài nguyên cũng tăng theo quy mô.

3.2.5 Máy học

Apache Spark cung cấp một thư viện máy học với tên gọi MLlib. Các nhà khoa học dữ liệu sử dụng MLlib để chạy phân tích hồi quy, phân loại và các tác vụ máy học khác. Bạn cũng có thể đào tạo các mô hình máy học với dữ liệu phi cấu trúc và dữ liệu có cấu trúc cũng như triển khai chúng cho các ứng dụng kinh doanh.

Ngược lại, Apache Hadoop không có các thư viện máy học được tích hợp sẵn. Thay vào đó, bạn có thể tích hợp Spark với các phần mềm khác như Apache Mahout để xây dựng các hệ thống máy học. Việc lựa chọn phần mềm phụ thuộc vào yêu cầu cụ thể của khối lượng công việc. Bạn có thể cân nhắc các yếu tố như kích cỡ và độ phức tạp của dữ liệu, loại mô hình máy học bạn muốn sử dụng, cũng như các yêu cầu về hiệu năng và khả năng điều chỉnh quy mô của ứng dụng.

Phần 4. Tổng kết

4.1 Ưu điểm

Dễ sử dụng

Apache Spark có giao diện thân thiện, phù hợp cả với người dùng không chuyên nhờ API dễ tiếp cận cho Scala, Python, Java và R. So với Hadoop, Spark giảm độ phức tạp khi không yêu cầu lập trình MapReduce thủ công.

Tốc độ vượt trội

Spark xử lý dữ liệu thời gian thực nhanh chóng, đạt hiệu suất cao nhờ in-memory processing. Ứng dụng của Spark bao gồm phát hiện gian lận trong giao dịch ngân hàng và phân tích hàng triệu sự kiện mỗi giây, đảm bảo tính tin cậy cho các hệ thống quan trọng.

Hỗ trợ thư viện tích hợp

Spark hỗ trợ nhiều thư viện mạnh như Spark Streaming (xử lý dữ liệu thời gian thực), Spark SQL (truy vấn dữ liệu có cấu trúc), MLlib (máy học) và GraphX (xử lý đồ thị). Các thư viện này mở rộng khả năng ứng dụng từ phân tích dữ liệu đến học máy và mạng xã hội.

Khả năng tương thích cao

Spark tích hợp dễ dàng với nhiều hệ thống lưu trữ như HDFS, S3 và Cassandra, đồng thời hỗ trợ đa ngôn ngữ, đặc biệt là Python, giúp tối ưu hóa trong cộng đồng phân tích dữ liệu.

4.2 Nhược điểm

Một trong những nhược điểm quan trọng của Apache Spark là tài nguyên tiêu tốn lớn. Do việc duy trì các dữ liệu trong bộ nhớ (in-memory) để tối ưu hóa hiệu suất, Spark yêu cầu một lượng tài nguyên bộ nhớ đáng kể, đặc biệt là khi xử lý các tập dữ liệu lớn. Điều này có thể gây khó khăn trong việc quản lý và triển khai trên các hệ thống có tài nguyên hạn chế.

Spark Streaming, mặc dù có khả năng xử lý dữ liệu thời gian thực, nhưng vẫn tồn tại độ trễ. Độ trễ này phụ thuộc vào độ dài của các mini-batches, gây khó khăn cho các ứng dụng yêu cầu phản ứng ngay lập tức và có độ chính xác cao trong xử lý dữ liệu.

Một vấn đề khác của Apache Spark là sự phức tạp trong quản lý và xử lý dữ liệu không cấu trúc. Trong khi Spark tập trung chủ yếu vào dữ liệu có cấu trúc, việc xử lý dữ liệu không cấu trúc như văn bản hoặc JSON có thể đòi hỏi sự tốn kém và đôi khi không hiệu quả.

Cuối cùng, quản lý và giám sát một hệ thống Apache Spark phân tán cũng đòi hỏi kiến thức chuyên sâu và kỹ năng quản trị cao. Việc tối ưu hóa cấu hình, theo dõi hiệu suất, và xử lý lỗi có thể là nhiệm vụ phức tạp, đặc biệt là trong các môi trường lớn.

Phần 5. Tham khảo

Nguồn tài liệu, học liệu tham khảo:

[1] Apache Spark (2024), "Spark Documentation Overview and All Components and Architectures," *Apache Spark Documentation*. Truy cập ngày 25 tháng 10 năm 2024, từ https://spark.apache.org/docs/3.5.2/.

[2] Apache Hadoop (2024), "Hadoop Documentation Overview," *Apache Hadoop Documentation*. Truy cập ngày 26 tháng 10 năm 2024, từ https://hadoop.apache.org/docs/r3.4.0/.

[3] Wikipedia (2024), "Big Data and Its Applications in Life," *Wikipedia Tiếng Việt*. Truy cập ngày 26 tháng 10 năm 2024, từ https://vi.wikipedia.org/wiki/D%E1%BB%AF_li%E1%BB%87u_l%E1%BB%9Bn.

[4] Wikipedia (2024), "About Apache Spark and All Definitions," *Wikipedia*. Truy cập ngày 29 tháng 10 năm 2024, từ https://en.wikipedia.org/wiki/Apache_Spark.

[5] Wikipedia (2024), "Apache Hadoop," *Wikipedia*. Truy cập ngày 30 tháng 10 năm 2024, từ https://en.wikipedia.org/wiki/Apache_Hadoop.

[6] Phúc Ngọc Nghĩa (2020), "Introduction to Apache Spark," *Viblo*. Truy cập ngày 31 tháng 10 năm 2024, từ https://viblo.asia/p/tim-hieu-ve-apache-spark-ByEZkQQW5Q0.

[7] Amazon Web Services (2024), "Comparison of Spark and Hadoop," *AWS Documentation*. Truy cập ngày 31 tháng 10 năm 2024, từ https://aws.amazon.com/vi/compare/the-difference-between-hadoop-vs-spark.

[8] ISB Insight (2024), "Big Data Processing Tools," *ISB Insight*. Truy cập ngày 31 tháng 10 năm 2024, từ https://insight.isb.edu.vn/top-8-cong-cu-big-data-ban-nen-biet/.

[9] De Manejar (2021), "Resilient Distributed Datasets," *De Manejar Blog*. Truy cập ngày 31 tháng 10 năm 2024, từ https://demanejar.github.io/posts/spark-rdd/.

[10] Nguyệt Linh (2024), "What is Apache Spark?" *VinaHost Blog*. Truy cập ngày 31 tháng 10 năm 2024, từ https://vinahost.vn/apache-spark-la-gi/.

[11] Jules S. Damji (2020), *Learning Spark 2.0*, O'Reilly Media.