**Data Analytics with Spark Using Python**

# **Preface**

* Spark is at the heart of the disruptive Big Data and open source software revolution
* The interest in and use of Spark have grown exponentially, with no signs of abating

## **The focus of the Book**

* The fundamentals of the Spark project (including the runtime, cluster, and application architecture)
* The Python API for Spark (PySpark)
* The scope of this book is quite broad, covering aspects of Spark from core Spark programming to Spark SQL, Spark Streaming, machine learning, and more

## **How to Use This Book**

|  |  |
| --- | --- |
| Part 1: Spark Foundations | Chapter 1 Introducing Big Data, Hadoop, and Spark |
| Chapter 2 Deploying Spark |
| Chapter 3 Understanding the Spark Cluster Architecture |
| Chapter 4 Learning Spark Programming Basics |
| Part 2: Beyond the basic | Chapter 5 Advanced Programming Using the Spark Core |
| Chapter 6 SQL and NoSQL Programming with Spark |
| Chapter 7 Stream Processing and Messaging Using Spark |
| Chapter 8 Introduction to Data Science and Machine Learning Using Spark |

## **About the Author**

* Jeffrey Aven is an independent Big Data, open-source software, and cloud computing professional based out of Melbourne, Australia.
* Jeffrey is a highly regarded consultant and instructor and has authored several other books including Teach Yourself Apache Spark in 24 Hours and Teach Yourself Hadoop in 24 Hours

*Sample data and source code for each of the exercises in this book, which is available at http://sparkusingpython.com. You can also view or clone the GitHub repository for this book at https://github.com/sparktraining/spark\_using\_python.*

# **Chapter 2 – Deploying Spark**

|  |  |
| --- | --- |
| In This Chapter:   * Overview of the different Spark deployment modes * How to install Spark * The contents of a Spark installation * Overview of the various methods available for deploying Spark in the cloud   Chương này trình bày những kiến ​​thức cơ bản về cách triển khai Spark, cách cài đặt Spark và cách khởi động và chạy các cụm Spark.  Nó thảo luận về các chế độ triển khai và bộ lập lịch khác nhau có sẵn cho các cụm Spark, cũng như các tùy chọn để triển khai Spark trên đám mây. |  |

Apache Spark là một khung máy tính phân tán có hỗ trợ tích hợp để xử lý hàng loạt và luồng dữ liệu lớn, hầu hết quá trình xử lý đó xảy ra trong bộ nhớ mang lại hiệu suất tốt hơn. Nó có các mô-đun tích hợp cho SQL, học máy, xử lý đồ thị, v.v.

## **Spark Deployment Modes**

|  |  |
| --- | --- |
| Local mode | * Local mode cho phép tất cả các quy trình Spark chạy trên một máy duy nhất, tùy chọn sử dụng bất kỳ số core nào trên hệ thống cục bộ. * Sử dụng Local mode thường là một cách nhanh chóng để kiểm tra cài đặt Spark mới và nó cho phép nhanh chóng kiểm tra các quy trình Spark dựa trên các tập dữ liệu nhỏ. * Khi chạy Spark Local mode, bạn có thể truy cập bất kỳ dữ liệu nào trên local filesystem. * Mặc dù Local mode cho phép bạn thiết lập và chạy nhanh chóng, nhưng nó bị hạn chế về khả năng mở rộng và hiệu quả đối với các trường hợp sử dụng sản xuất. |
| Spark Standalone | * Spark Standalone đề cập đến bộ lập lịch tích hợp hoặc "độc lập" * Nhiều quy trình hoặc dịch vụ máy chủ lưu trữ, chạy trong một cụm Spark Standalone và mỗi dịch vụ đóng một vai trò trong việc lập kế hoạch, điều phối và quản lý một ứng dụng Spark nhất định đang chạy trên cụm      * Với Spark Standalone, có thể bắt đầu và vận hành nhanh chóng mà không cần phải phụ thuộc hoặc cân nhắc về môi trường. * Mỗi bản phát hành Spark bao gồm mọi thứ cần để bắt đầu, bao gồm tệp nhị phân và tệp cấu hình cho bất kỳ máy chủ lưu trữ nào để đảm nhận bất kỳ vai trò cụ thể nào trong một cụm Spark Standalone. |
| Spark on YARN (Hadoop) | * Phương pháp triển khai phổ biến nhất cho Spark là sử dụng khung quản lý tài nguyên YARN được cung cấp cùng với Hadoop. YARN là thành phần cốt lõi của Hadoop cho phép bạn lập lịch và quản lý khối lượng công việc trên một cụm Hadoop. * Các quy trình Spark như Driver, Master, and Executors được lưu trữ hoặc hỗ trợ bởi các quy trình YARN (Resource Manager, Node Manager, and Application Master) * Có hai chế độ triển khai cụm khi sử dụng YARN làm bộ lập lịch: cluster and client |
| Spark on Mesos | * Apache Mesos là open source cluster manager developed được phát triển tại Đại học California, Berkeley * Mesos có khả năng lập lịch các loại ứng dụng khác nhau, cung cấp chia sẻ tài nguyên chi tiết giúp sử dụng cụm hiệu quả hơn. |

## **Preparing to Install Spark**

### Getting Spark

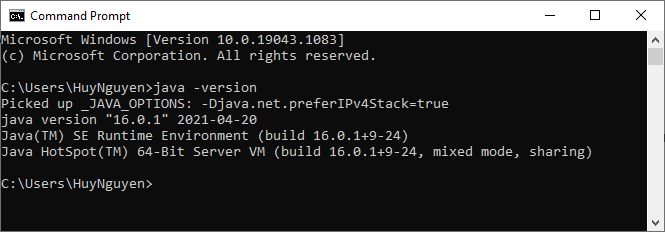
* Using a Spark release is often the easiest way to install Spark on a given system. Spark releases are downloadable from <http://spark.apache.org/downloads.html>.
* These releases are cross-platform: They target a JVM environment, which is platform agnostic.

|  |  |  |
| --- | --- | --- |
| * Linux * Windows * Mac OS X | * 8 GB RAM and more * 8 or more CPU core * 10 GB or greater network speed * Sufficient local disk space for storage | * Java (JDK preferably) (8./11. ~~16.~~) * Python/R |

### Installing Spark on Linux or Mac OS X

### Installing Spark on Windows

1. Download và cài đặt Java

* Download và cài đặt JDK  [tại đây](https://www.oracle.com/technetwork/java/javase/downloads/index.html)
* Thiết lập biến môi trường JAVA\_HOME với đường dẫn tới thư mục jdk (ví dụ: C:\Program Files\Java\jdk1.8.0\_191) trong Enviroment Variable (vào Control Panel, chọn System rồi chọn Advanced System settings ở góc bên góc trái)
* Thêm thư mục bin của jdk (%JAVA\_HOME%\bin) vào trong biến môi trường Path.
* Lưu lại các thiết lập rồi mở cửa sổ command để kiểm tra bằng lệnh java -version.

2. Download và cài đặt Scala/Python

* Download và cài đặt Scala [tại đây](https://www.scala-lang.org/download/) (kéo xuống chọn Binary for windows)
* Thiết lập biến môi trường SCALA\_HOME và Path tương tự như phần cài đặt Java
* Kiểm tra thiết lập bằng lệnh scala -version.

3. Download Spark

* Download Spark [tại đây](https://spark.apache.org/downloads.html) rồi tiến hành tiến hành giải nén file
* Thiết lập biến môi trường SPARK\_HOME và Path tương tự như trên

4. Download Winutils

* + Tiến hành download Winutils [tại đây](https://github.com/steveloughran/winutils), chú ý lựa trọn phiên bản Hadoop phù hợp với phiên bản đã lựa trọn trong phần download Spark
  + Click vào phiên bản Hadoop phù hợp, kéo xuống bên dưới để download file winutils.exe
  + Lưu file winutils.exe vào một thư mục bất kỳ và tạo biến môi trường HADOOP\_HOME với đường dẫn tới thư mục đó (nếu khi chạy spark bị lỗi không load được Winutils thì xem lại đường dẫn của Winutils vì có thể nó cần được để trong thư mục bin)

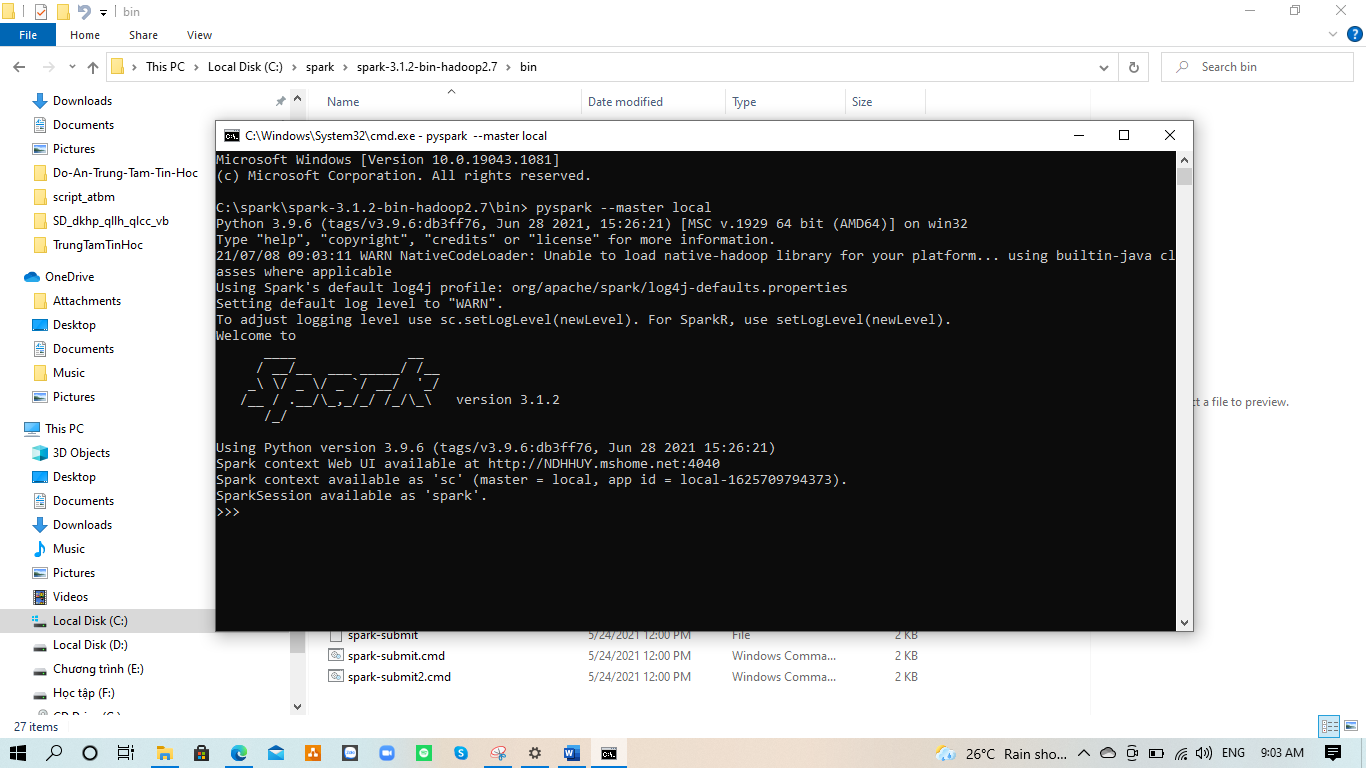
5. Thay đổi quyền truy cập cho thư mục tmp/hive

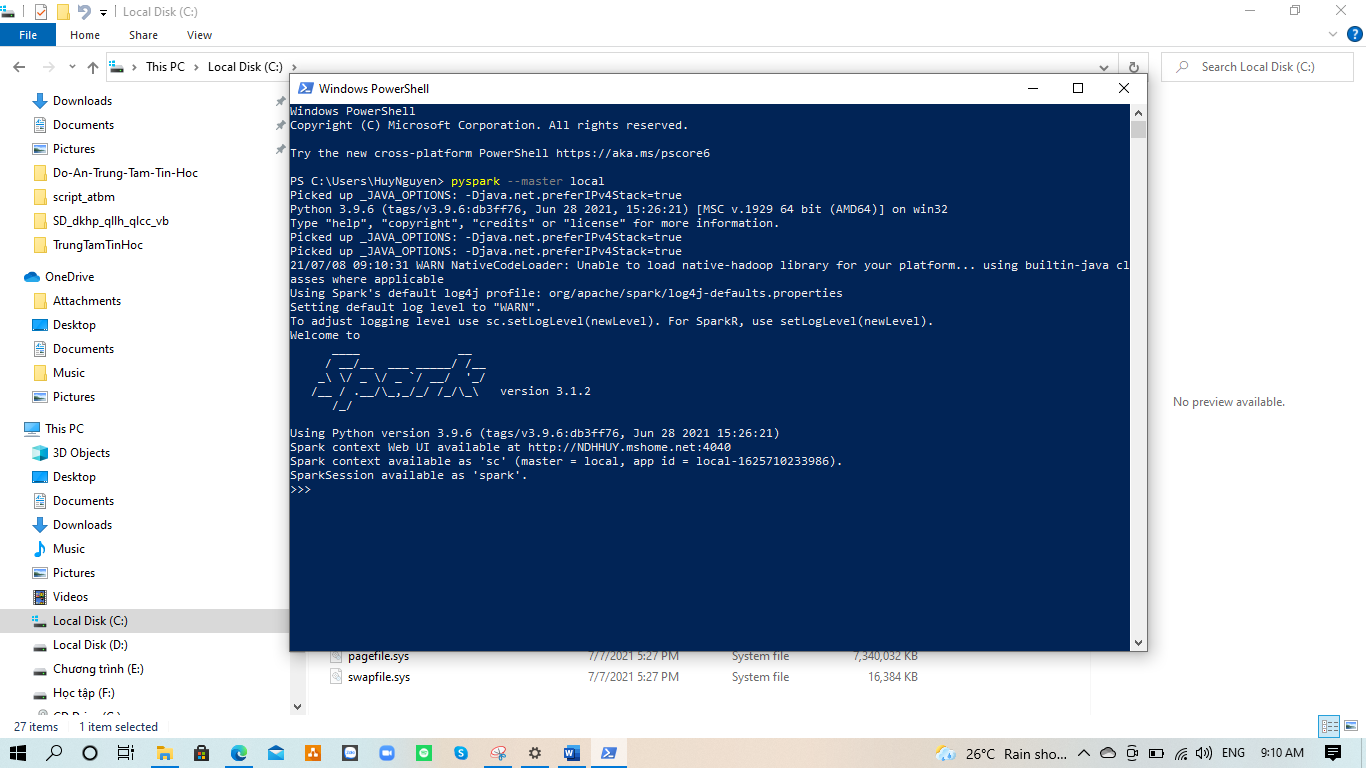
* + Sau khi hoàn thành các bước trên, sẽ có thư mục tmp/hive được tạo ra trong ổ C. Ta cần phải thay đổi quyền truy cập cho thư mục này nếu không khi chạy Spark sẽ báo lỗi
  + Để thực hiện việc này, ta mở một cửa số command rồi dùng Winutils để thay đổi permission như sau (chmod 777: cấp quyền read, write và execute

|  |  |
| --- | --- |
| 1 | winutils.exe chmod 777 C:\tmp\hive |

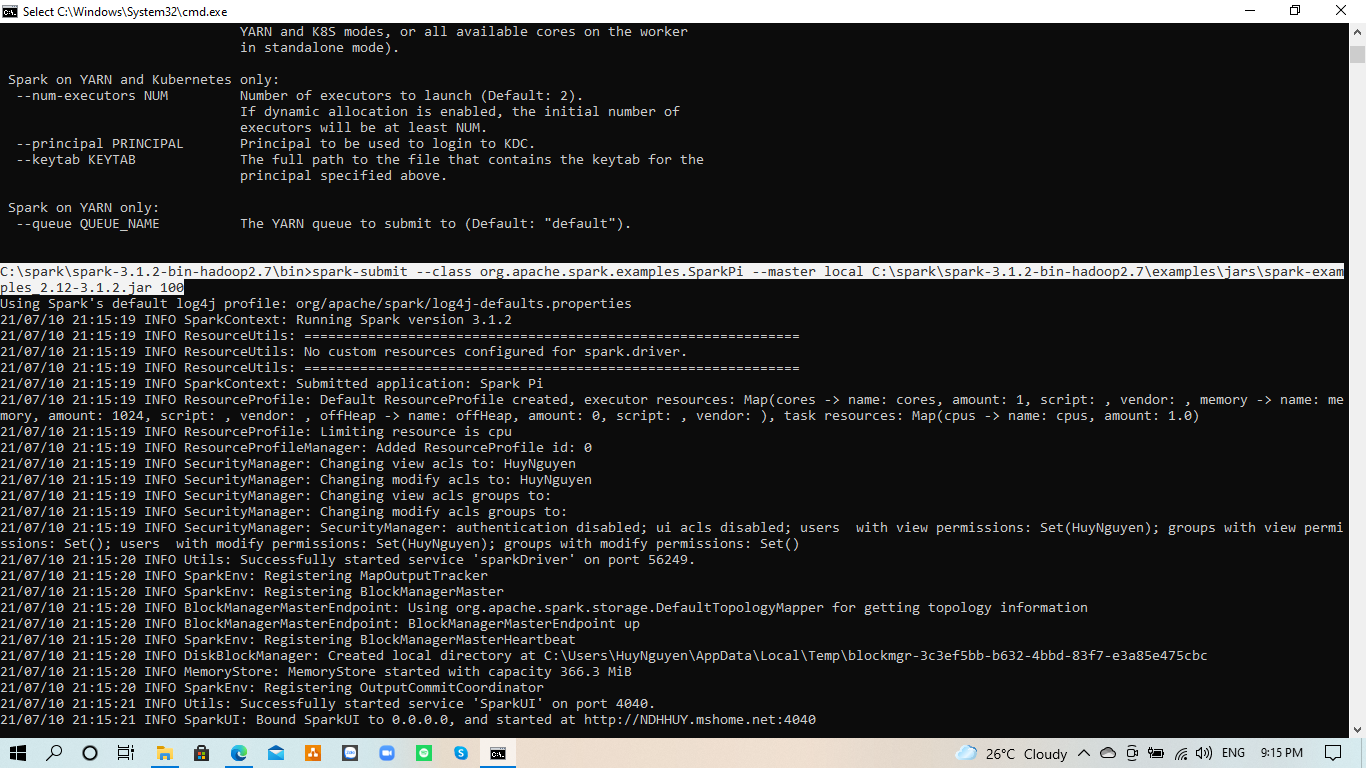
6. Kiểm tra hoàn thiện cài đặt

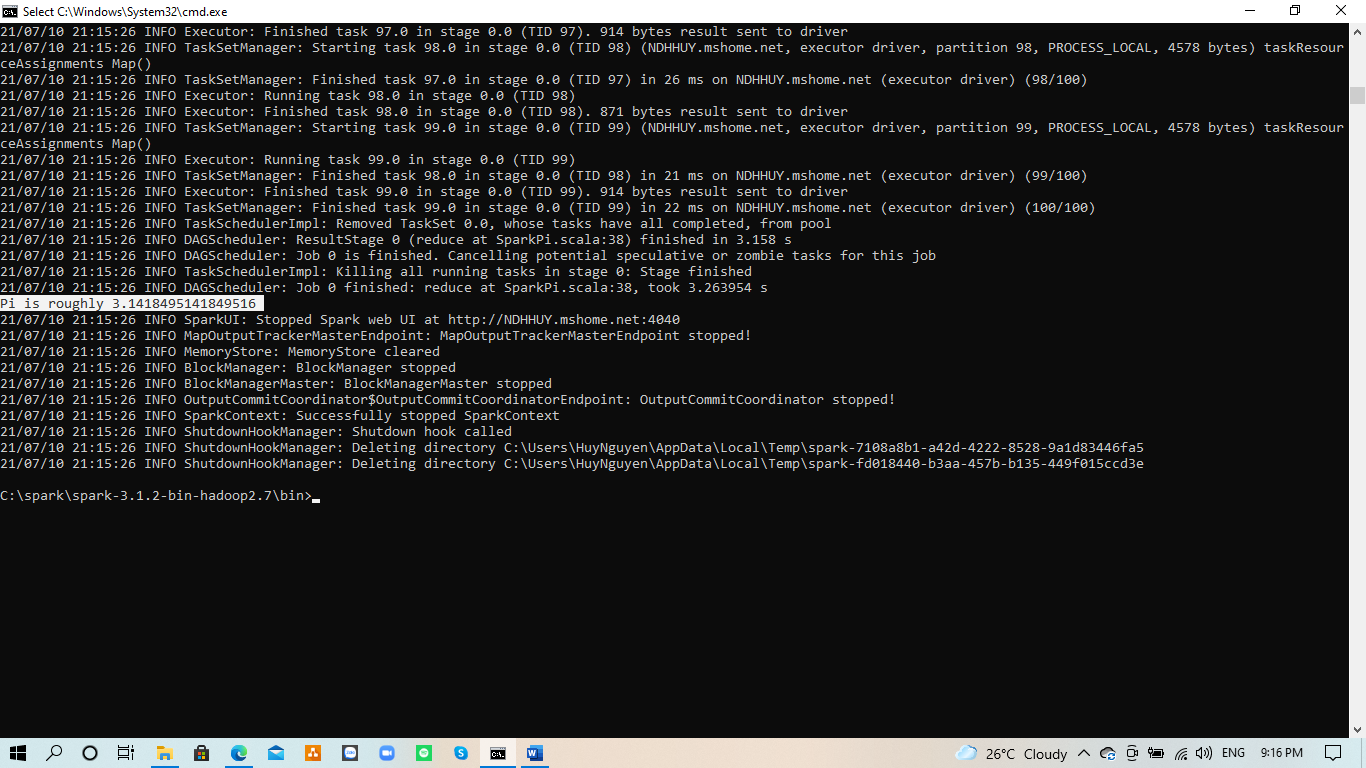
* Chạy thử một chương trình Spark nhỏ để kiểm tra xem việc cài đặt đã hoàn tất hay chưa.
* Dùng lệnh cmd để mở một của sổ command.
* Gõ lệnh spark-shell
* Gõ một chương trình nhỏ để chạy thử val





spark-submit --class org.apache.spark.examples.SparkPi --master local C:\spark\spark-3.1.2-bin-hadoop2.7\examples\jars\spark-examples\_2.12-3.1.2.jar 100





[PySpark with jupyter notebook | Install Apache Spark , PySpark, Java, Python and Anaconda on windows - YouTube](https://www.youtube.com/watch?v=DznteGdeJoA)

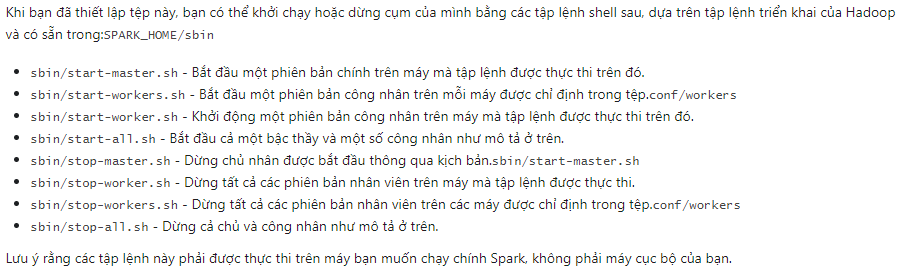
### Exploring the Spark Installation

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | bin/ | Chứa tất cả các lệnh / tập lệnh để chạy các ứng dụng Spark tương tác thông qua các chương trình shell như pyspark, spark-shell, spark-sql và sparkR hoặc ở chế độ hàng loạt bằng cách sử dụng spark-submit |
| conf/ | Chứa các mẫu cho các tệp cấu hình Spark mà bạn có thể sử dụng để đặt các giá trị cấu hình Spark (spark-defaults.conf.template), cũng như tập lệnh shell được sử dụng để đặt các biến môi trường cần thiết cho các quy trình Spark (spark-env.sh.template). Ngoài ra còn có cấu hìnhcác mẫu để kiểm soát việc ghi nhật ký (log4j.properties.template), một bộ sưu tập số liệu (metrics.properties.template) và một mẫu cho tệp nô lệ (slave.template), kiểm soát tệp nàocác nút nô lệ có thể tham gia cụm Spark đang chạy ở chế độ Độc lập. |
| data/ | Chứa tập dữ liệu mẫu được sử dụng để kiểm tra mllib, graphx,và phát trực tuyến các thư viện trong dự án Spark (tất cả đềusẽ được thảo luận sau trong cuốn sách này). |
| example/ | Chứa mã nguồn và các tập hợp đã biên dịch (tệp jar) cho tất cả các ví dụ đi kèm với bản phát hành Spark, bao gồm cả ứng dụng Công cụ ước tính Pi được sử dụng trong các ví dụ trước. Các chương trình mẫu được bao gồm trong Java, Python, R và Scala. Bạn cũng có thể tìm thấy mã mới nhất cho các ví dụ được bao gồm tại <https://github.com/apache/spark/tree/master/examples>. |
| jars/ | Chứa các cụm chính cho Spark cũng như các cụm cho các dịch vụ hỗ trợ được Spark sử dụng, chẳng hạn như snappy, py4j, parquet, và hơn thế nữa. Thư mục này được bao gồm trong CLASSPATH cho Spark theo mặc định |
| kubernetes/ |  |
| licenses/ | Bao gồm các tệp giấy phép bao gồm các dự án bao gồm khác, chẳng hạn như Scala và JQuery. Các tệp này dành cho mục đích tuân thủ pháp luật duy nhất và không bắt buộc phải chạy Spark |
| python/ | Chứa tất cả các thư viện Python cần thiết để chạy PySpark. Bạnnói chung không cần phải truy cập trực tiếp vào các tệp này. |
| R/ | Chứa gói SparkR và các thư viện liên quan và tài liệu. 🡪 Chapter 8 |
| sbin/ | Chứa các tập lệnh quản trị để bắt đầu và dừng master và slave dịch vụ cho các cụm Spark chạy ở chế độ Độc lập, cục bộhoặc từ xa, cũng như bắt đầu các quy trình liên quan đến YARN và Mesos. Bạn sẽ sử dụng một số đoạn mã này trong phần tiếp theokhi bạn triển khai một cụm nhiều nút ở chế độ Độc lập. |
| yarn/ | Chứa các thư viện hỗ trợ cho các ứng dụng Spark đang chạy trênYARN. Điều này bao gồm dịch vụ xáo trộn, một dịch vụ hỗ trợSpark sử dụng để di chuyển dữ liệu giữa các quy trình trong một cụm YARN. |

### Deploying a Multi-Node Spark Standalone Cluster

Hiện tại, Apache Spark supports Standalone, Apache Mesos, YARN và Kubernetes với tư cách là người quản lý tài nguyên. Standalone là trình quản lý tài nguyên của spark, dễ thiết lập có thể được sử dụng để bắt đầu mọi thứ nhanh chóng.

Có rất nhiều bài viết và đủ thông tin về cách bắt đầu một cụm độc lập trên môi trường Linux. Nhưng, không có nhiều thông tin về việc bắt đầu một cụm độc lập trên Windows.



Bây giờ bạn đã cài đặt và thử nghiệm cài đặt Spark ở chế độ Cục bộ, nó thời gian để giải phóng sức mạnh thực sự của Spark bằng cách tạo ra một Spark được phân phối đầy đủ cụm. Đối với bài tập này, bạn sẽ sử dụng bốn máy chủ Linux để tạo một ba-cụm nút bằng cách sử dụng bộ lập lịch biểu độc lập. Làm theo các bước sau:

1. Lập kế hoạch cấu trúc liên kết cụm và cài đặt Spark trên nhiều hệ thống. Bởi vì đây là một hệ thống phân tán, bạn cần cài đặt Spark, như được hiển thị trong các bài tập trước, trên ba máy chủ bổ sung. Ngoài ra, bạn cần phải chỉ định một máy chủ là Spark Master và máy chủ khác là Người lao động. Đối với bài tập này, máy chủ đầu tiên được đặt tên là sparkmaster và các máy chủ bổ sung là các tên sparkworker1, sparkworker2 và sparkworker3.

2. Cấu hình mạng. Tất cả các nút trong cụm Spark cần giao tiếp tất cả các máy chủ khác trong cụm. Cách dễ nhất để thực hiện điều này là bằng cách sử dụng tệp máy chủ (mục nhập cho tất cả máy chủ trong / etc / hosts trên mỗi hệ thống). Đảm bảo rằng mỗi nút có thể giải quyết nút kia. Lệnh ping có thể được sử dụng cho điều này; ví dụ: đây là cách bạn sử dụng nó từ máy chủ sparkmaster:

$ ping sparkworker1

3. Tạo và chỉnh sửa tệp spark-defaults.conf trên mỗi máy chủ. Để tạo và định cấu hình tệp spark-defaults.conf trên mỗi nút, chạy các lệnh sau trên máy chủ sparkmaster và sparkworker:

$ cd $ SPARK\_HOME / conf

$ sudo cp spark-defaults.conf.template spark-defaults.conf

$ sudo sed -i "\ $ aspark.master \ tspark: // sparkmaster: 7077" spark-

defaults.conf

4. Tạo và chỉnh sửa tệp spark-env.sh trên mỗi máy chủ. Để tạo và định cấu hình tệp spark-env.sh trên mỗi nút, hoàn thành các bước sau nhiệm vụ trên máy chủ sparkmaster và sparkworker:

$ cd $ SPARK\_HOME / conf

$ sudo cp spark-env.sh.template spark-env.sh

$ sudo sed -i "\ $ aSPARK\_MASTER\_IP = sparkmaster" spark-env.sh

5. Khởi động Spark Master. Trên máy chủ sparkmaster, hãy chạy như sau chỉ huy:

$ sudo $ SPARK\_HOME / sbin / start-master.sh

Kiểm tra quy trình Spark Master bằng cách xem giao diện người dùng web Spark Master tại

http: // sparkmaster: 8080 / .

6. Khởi động Công nhân Tia lửa. Trên mỗi nút sparkworker, hãy chạy như sau chỉ huy:

$ sudo $ SPARK\_HOME / sbin / start-slave.sh spark: // sparkmaster: 7077

Kiểm tra giao diện người dùng Spark Worker trên http: // sparkslaveN: 8081 / .

7. Kiểm tra cụm nhiều nút. Chạy ví dụ về Công cụ ước tính Pi được tích hợp sẵn từ đầu cuối của bất kỳ nút nào trong cụm:

$ spark-submit --class org.apache.spark.examples.SparkPi \

--master spark: // sparkmaster: 7077 \

--driver-memory 512m \

--executor-memory 512m \

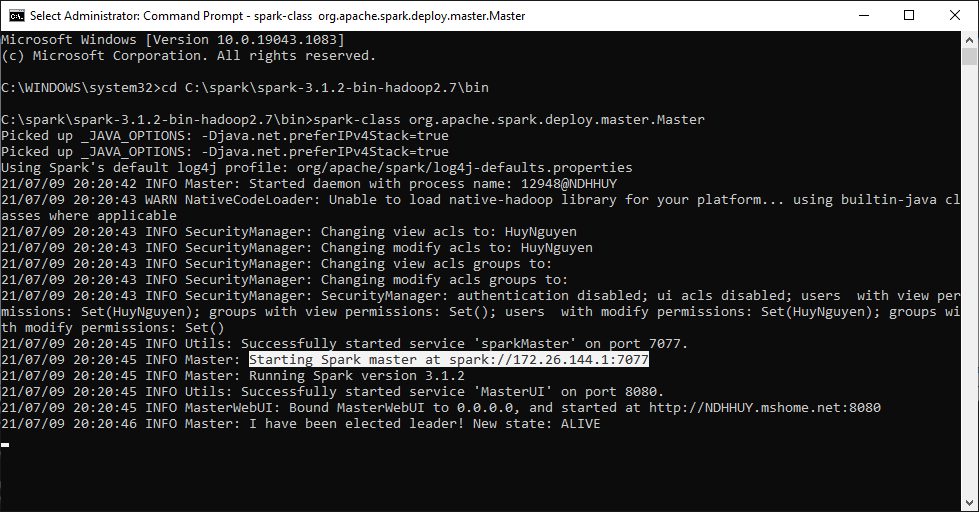
--executor-core 1 \

$ SPARK\_HOME / ví dụ / lọ / ví dụ tia lửa \* .jar 10000

Bạn sẽ thấy đầu ra tương tự như kết quả từ các bài tập trước. Bạn cũng có thể bật SSH không mật khẩu (Secure Shell) cho Spark Master để workers Yarn. Điều này là bắt buộc để kích hoạt đăng nhập từ xa cho daemon nô lệ các hành động khởi động và tắt máy.

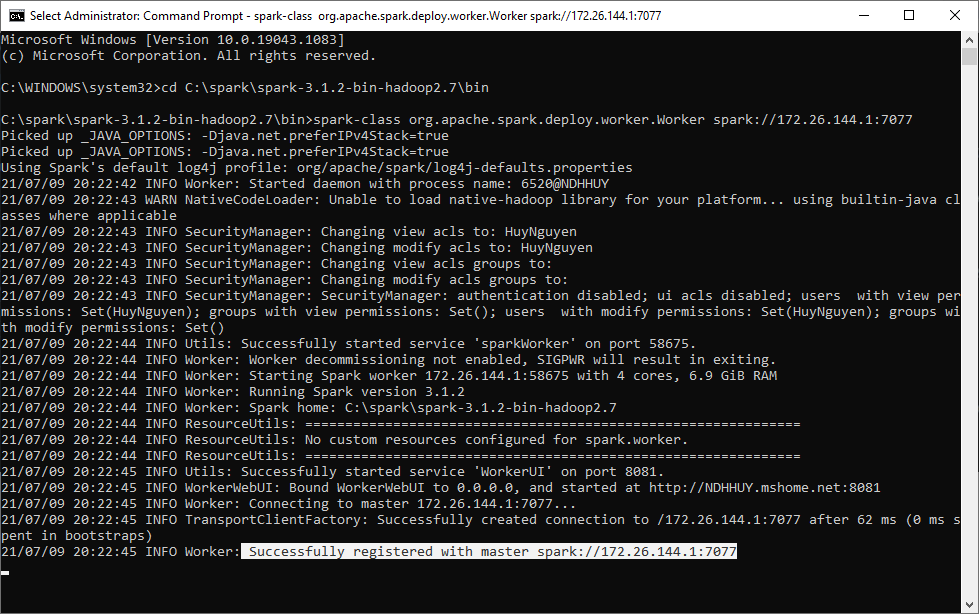
**Set up Master Node**

spark-class org.apache.spark.deploy.master.Master

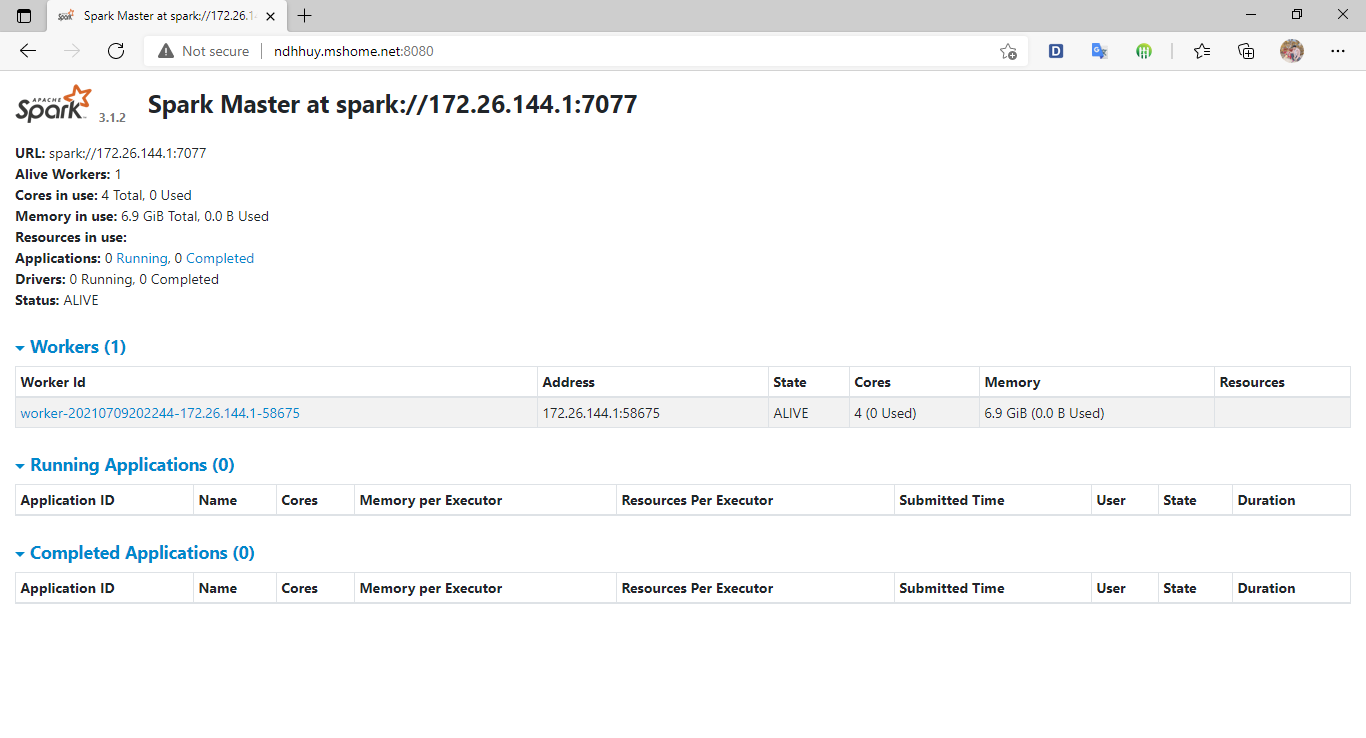


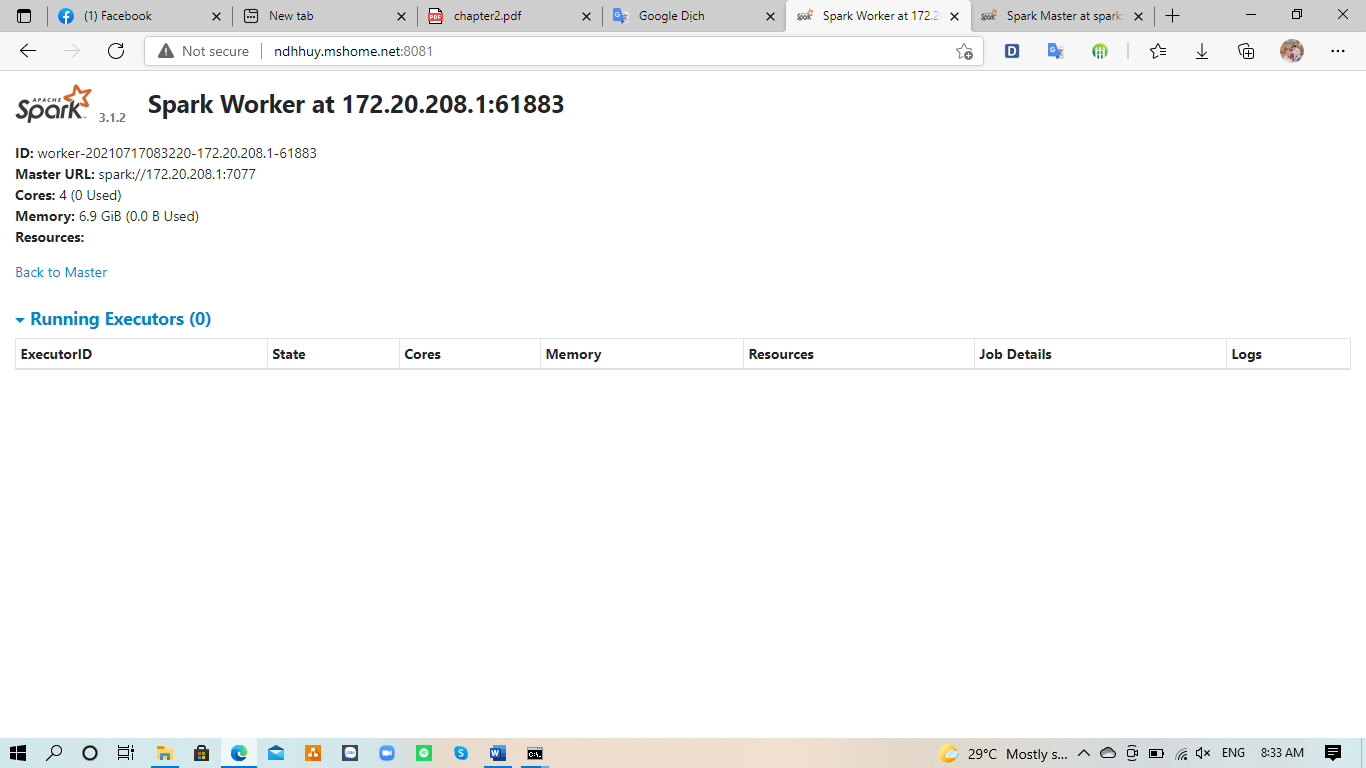
**Set up Worker Node**

spark-class org.apache.spark.deploy.worker.Worker spark://172.26.144.1:7077



**Spark UI**





[Apache Spark standalone cluster on Windows | by Amar Gajbhiye | Medium](https://aamargajbhiye.medium.com/apache-spark-setup-a-multi-node-standalone-cluster-on-windows-63d413296971)

### Deploying Spark in the Cloud

* Software-as-a-Service (SaaS)
* Infrastructure-as-a-Service (IaaS) (Cơ sở hạ tầng như dịch vụ)
* Platform-as-a-Service (PaaS) (Nền tảng như dịch vụ)

#### Amazon Web Services (AWS)

* Amazon Web Services (AWS) là nền tảng đám mây toàn diện và được sử dụng rộng rãi nhất, cung cấp trên 200 dịch vụ đầy đủ tính năng từ các trung tâm dữ liệu trên toàn thế giới. Hàng triệu khách hàng, bao gồm các công ty khởi nghiệp tăng trưởng nhanh nhất, các tập đoàn lớn nhất cũng như các cơ quan hàng đầu của chính phủ, đều tin tưởng vào AWS để giảm chi phí, trở nên linh hoạt hơn và đổi mới nhanh hơn.
* Đầy đủ chức năng: từ các công nghệ cơ sở hạ tầng như máy tính, ổ lưu trữ và cơ sở dữ liệu, đến các công nghệ mới nổi, như machine learning và trí tuệ nhân tạo, kho dữ liệu và phân tích, và Internet of Things
* Bảo mật nhất: được thiết kế để trở thành môi trường điện toán đám mây bảo mật và linh hoạt nhất hiện nay

#### Google Cloud Platform (GCP)

* Google Cloud hay còn gọi là Google Cloud Platform (GCP) chính là một nền tảng của kỹ thuật điện toán đám mây cho phép các cá nhân, tổ chức, các doanh nghiệp, các cơ quan có thể xây dựng, phát triển, và hoạt động các ứng dụng của mình trên hệ thống phần mềm do google tạo ra. Các ứng dụng rất phổ biến hiện nay được mọi người sử dụng rất nhiều như: Trình duyệt Chrome, ứng dụng bản đồ Google Map, Google Apps, kênh Youtube…
* Google Cloud cung cấp tất cả các giải pháp quản lý cho doanh nghiệp, giúp doanh nghiệp có thể phát triển hệ thống công nghệ của mình một cách chính xác, hiện đại. Bên cạnh đó, GC còn giúp người dùng và doanh nghiệp giải quyết các vấn đề như: Developer (phát triển), Management (Quản lý), Computer Engine, Mobile, Storage, Big Data…
* Một điểm khác biệt nữa mà GC mang lại so với các dịch vụ đám mây khác đó chính là hệ thống DataCenter luôn ổn định và có độ bảo mật dữ liệu cực cao, giúp bảo vệ dữ liệu người dùng và khách hàng trước sự dòm ngó và xâm nhập trái phép của các hacker công nghệ.

#### Databricks

* Azure Databricks là giải pháp phần mềm Big Data Analytics Software với chức năng và chi phí phù hợp cho các doanh nghiệp từ nhỏ và vừa tới các doanh nghiệp lớn.
* Phần mềm Azure Databricks được đánh giá cao bởi cả người dùng lẫn chuyên gia trong lĩnh vực Database Software.

### Summary

# **Chapter 4 – Learning Spark Programming Basics**

|  |  |
| --- | --- |
| In This Chapter:   * In This Chapter: * Resilient Distributed Datasets (RDDs) * How to load data into Spark RDDs * Transformation and actions on RDDs * How to perform operations on multiple RDDs |  |

## Giới thiệu 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).

* Resilent: RDD có khả năng phục hồi (if a node performing an operation in Spark is lost, the dataset can be reconstructed)
* Distributed: RDD cung cấp một dạng bộ nhớ chia sẻ hiệu quả để trao đổi dữ liệu giữa các tiến trình (Executors) trên các nút khác nhau (Workers)
* Dataset: RDD là tập dữ liệu bao gồm các bản ghi

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 chính 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 MapRedure** | **Thực thi trên Spark RDD** |
|  | 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. | 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. |
| **Iterative Operation** |  |  |
| **Interactive Operations** |  |  |

## Loading Data into RDDs

* Loading data from a file or files
* Loading data from a data source, such as a SQL or NoSQL datastore (Chapter 6)
* Loading data programmatically
* Loading data from a stream (Chapter 7)

### Tạo RDD từ File/Files

* Spark cung cấp các phương thức API để tạo RDD từ một tệp, các tệp hoặc nội dung của một thư mục.
* Các tệp có thể có nhiều định dạng khác nhau: không cấu trúc, bán cấu trúc (JSON,…), cấu trúc (CSV,..).
* Spark cũng hỗ trợ một số định dạng mã hóa nhị phân được tuần tự hóa phổ biến, chẳng hạn như SequenceFiles và bộ đệm giao thức (protobufs), cũng như các định dạng tệp cộtchẳng hạn như Parquet và ORC .

### Phương pháp tạo RDD từ Text File/Files

#### textFile()

|  |  |  |
| --- | --- | --- |
| sc.textFile(name, minPartitions=None, use\_unicode=True)   * Được sử dụng để tạo RDD từ các tệp (được nén hoặc không nén), thư mục hoặc glob patterns (file patterns with wildcards). * Khi đọc tất cả các tệp trong một thư mục bằng phương thức textFile (), mỗi dòng của mỗi tệptệp đại diện cho một bản ghi riêng biệt không có ngữ cảnh về nguồn gốc tệp của dòng. | | |
| name | minPartitions | use\_unicode |
| chỉ định đường dẫn hoặc toàn cầu được tham chiếu, bao gồmlược đồ hệ thống tập tin | xác định số lượng phân vùng cần tạo. | chỉ định sử dụng Unicode hay UTF-8 làmlược đồ mã hóa ký tự |
|  | | |

#### wholeTextFiles()

|  |
| --- |
| sc.wholeTextFiles(path, minPartitions=None, use\_unicode=True)   * Cho phép bạn đọc một thư mục chứanhiều tệp * Mỗi tệp được biểu diễn dưới dạng một bản ghi bao gồm một khóa chứatên tệp và một giá trị chứa nội dung của tệp. |
|  |

### Tạo RDD từ Object File

Spark hỗ trợ một số triển khai object file phổ biến. Object file đề cập đến cấu trúc dữ liệu được tuần tự hóa mà con người thường không thể đọc được và được thiết kế để cung cấp cấu trúc và ngữ cảnh cho dữ liệu, giúp truy cập vào dữ liệu hiệu quả hơn cho nền tảng yêu cầu.

Sequence file là file tuần tự được mã hóa thường được sử dụng trong Hadoop. Tạo RDD bằng cách sử dụng phương thức sequenceFile()/ hadoopFile() cũng tương tự.

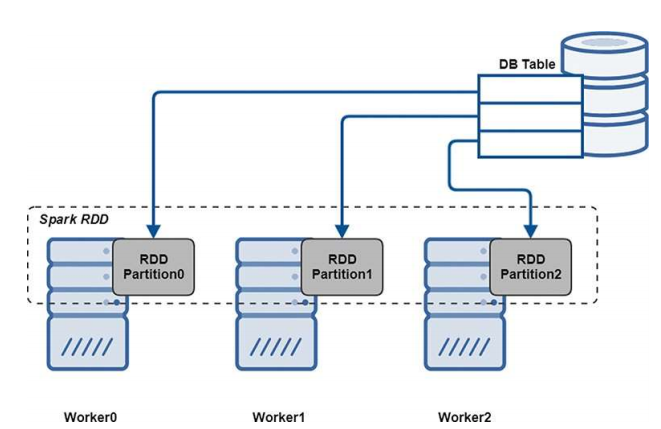
Ngoài ra, có hỗ trợ đọc và ghi các tệp Pickle, một định dạng tuần tự hóa đặc biệt cho Python. Đối tượng Java được tuần tự hóa với phương thức objectFile().

### Tạo RDD từ Data source

Dữ liệu này có thể đến từ nhiều hệ thống máy chủ và nền tảng cơ sở dữ liệu (Oracle, MySQL, Postgres và SQL Server...)

RDD được tạo bằng cách sử dụng dữ liệu từ cơ sở dữ liệu bên ngoài và cố gắng di chuyển dữ liệu thành nhiều phân vùng trên nhiều Worker. 🡪 tối đa hóa song song trong quá trình xử lý, đặc biệt là trong các giai đoạn ban đầu.

Ngoài ra, nếu chia bảng, thường theo không gian chính, thành các phân vùng khác nhau, các phân vùng cũng có thể tải song song và mỗi phân vùng chịu trách nhiệm về tìm nạp một tập hợp các hàng duy nhất.



* Java Database Connectivity (JDBC)
* read.jdbc()

### Tạo RDD từ Json File

JSON (JavaScript Object Notation) là một định dạng trao đổi dữ liệu phổ biến. JSON là một định dạng "tự mô tả", con người có thể đọc được và thường được sử dụng để trả lại phản hồi từ các dịch vụ web và RESTful APIs (một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web (thiết kế Web services) để tiện cho việc quản lý các resource..). Các đối tượng JSON được xử lý dưới dạng nguồn dữ liệu và được truy cập bằng phương thức read.json() được hiển thị thông qua SparkSession entry point.

|  |  |
| --- | --- |
| spark.read.json(path, schema=None) | path: chỉ định đường dẫn đầy đủ đến tệp JSON mà bạn đang sử dụng làmnguồn dữ liệu.  schema: tùy chọn để chỉ định lược đồ mục tiêu để tạo DataFrame. |
|  | |

spark.read.json(path, schema=None)

### Tạo RDD Programatically

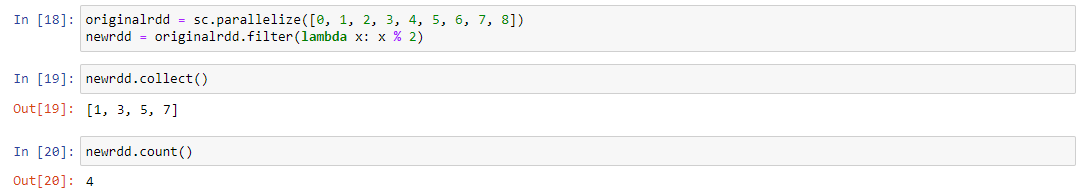
## Operations on RDDs

### Key RDD Concepts

#### Coarse-Grained Versus Fine-Grained Transformations

|  |  |
| --- | --- |
| Coarse-Grained | Hoạt động được thực hiện dựa trên RDD được coi là Coarse-Grained vì chúng áp dụng một chức năng (vd: map(), filter(),…) đối với mọi phần tử trong tập dữ liệu và chúng trả về tập dữ liệu với các phép biến đổi được áp dụng. |
| Fine-Grained | Thao tác với một bản ghi duy nhất hoặc ô dữ liệu, chẳng hạn như cập nhật một hàng trong cơ sở dữ liệu quan hệ hoặc put operations trong cơ sở dữ liệu NoSQL. |

#### Transformations, Actions, and Lazy Evaluation



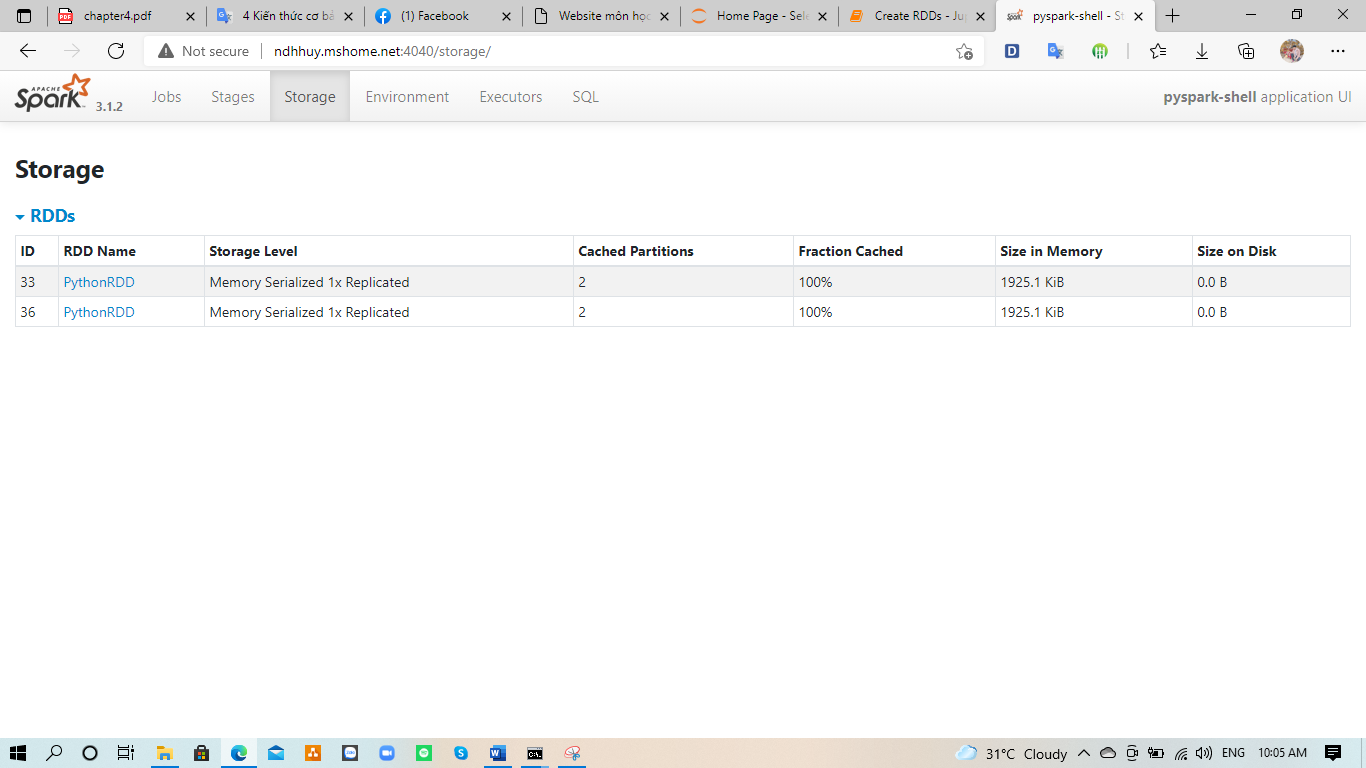
originalrdd bắt nguồn từ một bộ sưu tập các số được song song hóa. Cácchuyển đổi filter () sau đó được áp dụng cho từng phần tử trongoriginalrdd để bỏ qua các số chẵn trong bộ sưu tập. Sự biến đổi nàykết quả trong RDD được gọi là newrdd.Ngược lại với các phép biến đổi, trả về các đối tượng RDD mới, các hành động trả vềgiá trị hoặc dữ liệu cho chương trình điều khiển. Các hành động phổ biến bao gồm giảm (),collect (), count () và saveAsTextFile ().

Spark sử dụng đánh giá lười biếng, còn được gọi là thực thi lười biếng, trong việc xử lý Sparkcác chương trình. Đánh giá lười biếng ngăn cản quá trình xử lý cho đến khi một hành động được gọi (nghĩa làkhi yêu cầu đầu ra). Điều này dễ dàng được chứng minh bằng cách sử dụng một trình bao tương tác,nơi bạn có thể nhập một hoặc nhiều phương thức chuyển đổi thành RDD saukhác mà không bắt đầu xử lý. Thay vào đó, mỗi câu lệnh được phân tích cú pháp chochỉ tham chiếu cú ​​pháp và đối tượng. Sau khi yêu cầu một hành động như count ()hoặc saveAsTextFile (), một DAG được tạo cùng vớicác kế hoạch thực hiện. Sau đó, Trình điều khiển sắp xếp và quản lý các kế hoạch này trênNhững người thừa hành.Đánh giá lười biếng này cho phép Spark kết hợp các hoạt động nếu có thể, do đógiảm các công đoạn xử lý và giảm thiểu lượng dữ liệu được truyềngiữa những Người thực thi Spark trong một quá trình được gọi là xáo trộn.

#### RDD Persistence and Reuse (duy trì-tái sử dụng)

RDD được tạo ra và tồn tại chủ yếu trong bộ nhớ trên Executor. Theo mặc định, RDD là các đối tượng nhất thời chỉ tồn tại khi chúng được yêu cầu. Sau khi họchuyển đổi thành các RDD mới và không cần thiết cho bất kỳ hoạt động nào khác, chúngbị xóa vĩnh viễn. Điều này có thể có vấn đề nếu RDD được yêu cầu để biết thêmhơn một hành động vì nó phải được đánh giá lại toàn bộ mỗi lần. Mỗi tùy chọn để giải quyết vấn đề này là lưu vào bộ nhớ cache hoặc duy trì RDD bằng cách sử dụng persist()

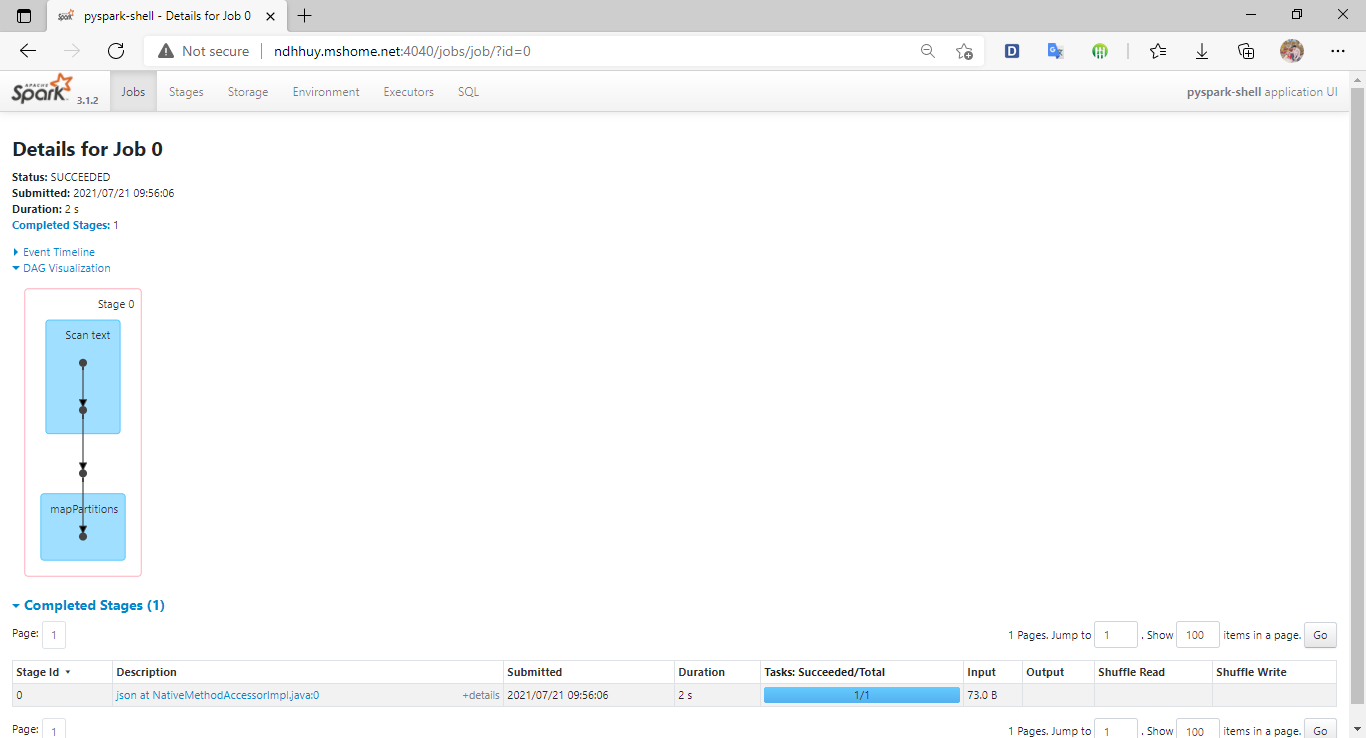
Sau khi yêu cầu duy trì RDD bằng cách sử dụng phương thức persist() (lưu ý rằng cũng có một phương thức cache () tương tự), RDD vẫn còn trong bộ nhớ trên tất cả các nút trong cụm nơi nó được tính sau khi hành động đầu tiên được gọi trên nó.Bạn có thể thấy RDD tồn tại trong giao diện người dùng ứng dụng Spark của mình trong tab Bộ nhớ,

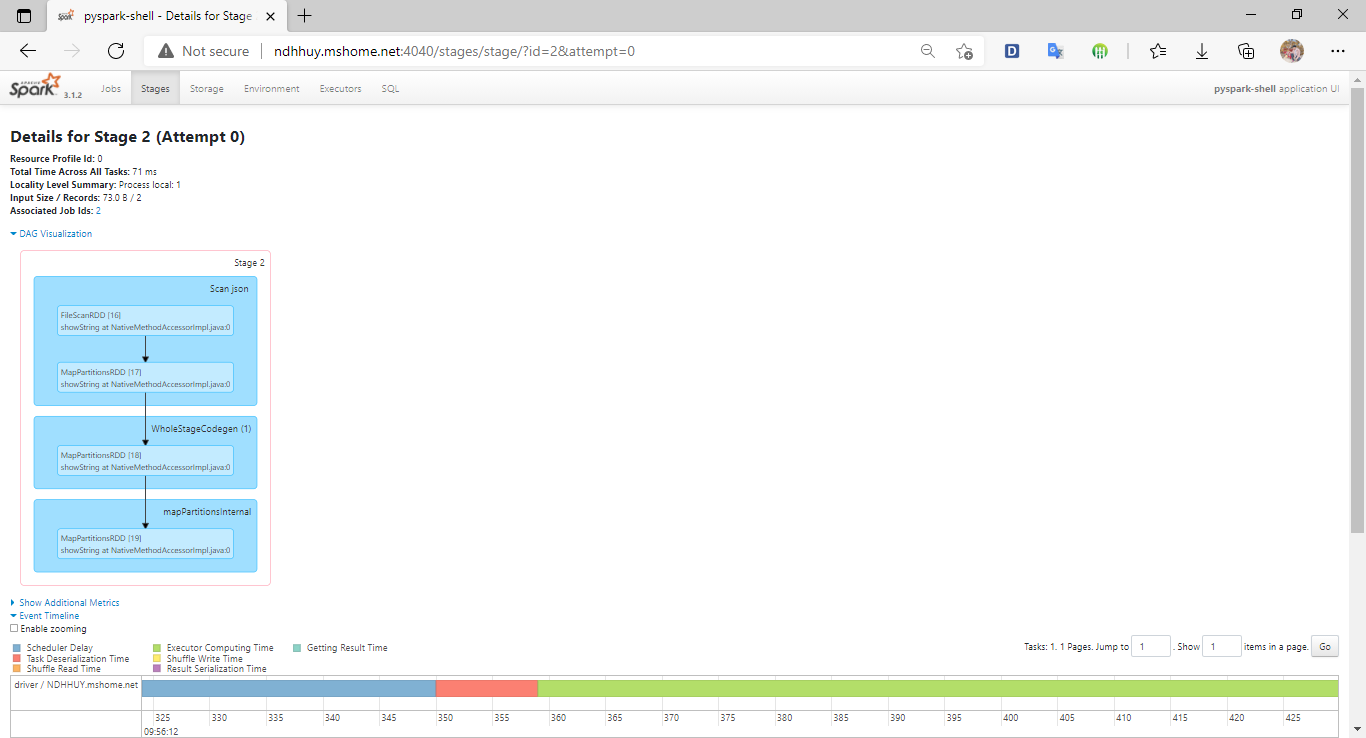


#### RDD Lineage

Khi một transformation(map hoặc filter etc) được gọi là, nó không được Spark thực thi ngay lập tức, thay vào đó một dòng dõi được tạo ra cho mỗi phép biến đổi. Một dòng dõi sẽ theo dõi những gì tất cả các biến đổi phải được áp dụng trên RDD đó, bao gồm cả vị trí từ nơi nó phải đọc dữ liệu.

Hành động được sử dụng để lưu kết quả vào một số vị trí hoặc để hiển thị nó. Thông tin dòng truyền tải RDD cũng có thể được in bằng cách sử dụng lệnh filteredRdd.toDebugString (filteredRdd là RDD ở đây). Ngoài ra, Trực quan hóa DAG hiển thị biểu đồ hoàn chỉnh một cách rất trực quan như sau:





Trong Spark, Lineage Graph là một biểu đồ phụ thuộc ở giữa RDD hiện có và RDD mới. Điều đó có nghĩa là tất cả các phụ thuộc giữa RDD sẽ được ghi lại trong biểu đồ, thay vì dữ liệu ban đầu.

#### Fault Tolerance with RDDs

Spark ghi lại lineage RDD, bao gồm lineage của tất cả các parent RDDs , parents’ parents,…. Bất kỳ RDD nào với tất cả các phân vùng của nó đều có thể được xây dựng lại về trạng thái ban đầu tại thời điểm xảy ra sự cố, có thể cóchẳng hạn như do lỗi nút. Bởi vì RDD được phân phối, chúngcó thể chịu đựng và phục hồi sau sự cố của bất kỳ nút nào.

Bạn có thể ngăn thời gian khôi phục dài đối với các hoạt động xử lý phức tạp bằng cách điểm kiểm tra hoặc lưu dữ liệu vào một đối tượng dựa trên tệp liên tục. (Chương 5,“Lập trình nâng cao sử dụng Spark Core API”)

#### Types of RDDs

* PairRDD: Một RDD của các cặp key/value. Bạn đã thấy loại này RDD vì nó được tạo tự động bằng cách sử dụng wholeTextFiles ().
* DoubleRDD: Một RDD chỉ bao gồm một tập hợp các giá trị kép. Vì các giá trị có cùng kiểu số, một số giá trị bổ sung các hàm thống kê có sẵn, bao gồm mean(), sum(), stdev(), variance(), histogram(), trong số những thứ khác.
* DataFrame (trước đây gọi là SchemaRDD): Một tập hợp phân tán dữ liệu được tổ chức thành các cột được đặt tên và đã nhập. DataFrame là tương đương với một bảng quan hệ trong Spark SQL. DataFrames bắt nguồn bằng các hàm read.jdbc () và read.json.
* SequenceFileRDD: Một RDD được tạo từ SequenceFile, nén hoặc không nén.
* HadoopRDD: Một RDD cung cấp chức năng cốt lõi để đọc dữ liệu được lưu trữ trong HDFS bằng API MapReduce v1.
* NewHadoopRDD: Một RDD cung cấp chức năng cốt lõi để đọc dữ liệu được lưu trữ trong Hadoop - ví dụ: tệp trong HDFS, nguồn trong HBase hoặc S3 - sử dụng API MapReduce mới (org.apache.hadoop.mapreduce).
* CoGroupedRDD: Một RDD nhóm cha mẹ của nó. Đối với mỗi khóa trongRDD mẹ, RDD kết quả chứa một bộ dữ liệu với danh sách các giá trị cho chìa khóa đó.
* JdbcRDD: Một RDD kết quả từ một truy vấn SQL đến một kết nối JDBC. Nó chỉ có sẵn trong API Scala.
* PartitionPruningRDD: Một RDD được sử dụng để cắt tỉa các phân vùng RDD hoặc các phân vùng để tránh khởi chạy tác vụ trên tất cả các phân vùng. Ví dụ, nếu bạn biết RDD được phân vùng theo phạm vi và DAG thực thi có bộ lọc trên khóa, bạn có thể tránh khởi chạy các tác vụ trên các phân vùng không cóphạm vi bao phủ phím.
* ShuffledRDD: RDD kết quả từ một lần xáo trộn, chẳng hạn như phân vùng lại dữ liệu.
* UnionRDD: Một RDD kết quả từ một phép toán union () chống lại hai hoặc nhiều RDD hơn.

### Basic RDD Transformations

Qua 1 phương thức transformations thì sẽ cho phép tạo mới 1 RDD từ 1 RDD đã tồn tại.

Tất cả các transformation đều là lazy, có nghĩa là các transformation này sẽ không thực hiện tính toán trong phương thức ngay mà chúng sẽ được lưu lại thành dữ liệu cơ bản( ví dụ như file) và chúng chỉ thực hiện tính toán khi 1 action được gọi để yêu cầu trả về kết quả cho driver program. Nhờ thiết kế này mà Spark chạy hiệu quả hơn.

Ví dụ:

Chúng ta có thể nhận thấy kết quả là 1 bộ dữ liệu được tạo qua hàm map và chỉ trả về kết quả cho driver program qua hàm reduce. Lazy ở đây tốt ở chỗ là chúng ta không phải tạo ra 1 tập dữ liệu và thực hiện lưu trữ chúng trong file rồi sau đó các actions sẽ phải tìm cách truy cập đến các file lưu trữ này mà thay vào đó, chúng ta sẽ tiết kiệm được thời gian(thời gian truy cập và thời gian lưu trữ) và không gian lưu trữ kết quả (RDD) qua transformation.

|  |  |
| --- | --- |
| Transformation | Ý nghĩa |
| map(*func*) | Trả về 1 RDD mới bằng cách truyền mỗi phần tử đầu vào (nguồn) qua hàm *func*. |
| filter(*func*) | Trả về 1 RDD mới bằng cách chọn những phần tử đầu vào (nguồn) mà hàm *func* trả về kết quả *true.* |
| flatMap(*func*) | Tương tự *map*nhưng khác *map* ở chỗ, mỗi phần tử đầu vào qua *flatMap*sẽ trả về 0 hoặc nhiều phần tử đầu ra(có thể hiểu qua *map*sẽ là 1-1). |
| mapPartitions(*func*) | Tương tự như *map* nhưng chạy riêng biệt trên mỗi vùng RDD,Hàm *func* phải có dạng Iterator[T] => Iterator[U] khi chạy RDD kiểuT. |
| mapPartitionsWithSplit(*func*) | Tương tự *mapPartitions*, nhưng hàm *func*được cung cấp thêm 1 sốnguyên để chỉ số split. |
| union(*otherDataset*) | Trả về 1 RDD mới là hợp của tập dữ liệu phần tử đầu vào(nguồn) vàcác phần tử của đối(*otherDataset*). |
| distinct([*numTasks*])) | Trả về 1 RDD mới chứa mỗi phần tử là duy nhất của tập dữ liệu nguồn(đầu vào). |
| groupBy() |  |
| groupByKey([*numTasks*]) | Khi gọi đến 1 tập dữ liệu (K,V) sẽ trả về 1 tập là cặp (K,Seq(V))( Tức lànhóm tập các phần tử cùng *Key*).  **Chú ý:**mặc định chỉ có 8 task song songkhi grouping. Có thể thay đổi số task song song này bằng việc truyềnvào tham số đầu vào. |
| reduceByKey(*func*, [*numTasks*]) | Khi gọi tập dữ liệu (K,V), trả về 1 tập (K,V) mà giá trị của key đượctổng hợp sử dụng hàm reduce *func.* |
| sortBy() |  |
| sortByKey([*ascending*], [*numTasks*]) | Khi gọi tập dữ liệu (K,V) với K có thể thực hiện sắp thứ tự được.Khi đó, nó sẽ trả về tập dữ liệu (K,V) được sắp sếp tăng dần hoặcgiảm dần theo key.  **Chú ý:***ascending*là kiểu Boolean. |
| join(*otherDataset*, [*numTasks*]) | Khi gọi tập dữ liệu có kiểu (K,V) và (K,W), nó sẽ trả về 1 cặp mới(K,(V,W))( nối 2 phần tử có cùng key). |
| cogroup(*otherDataset*, [*numTasks*]) | Khi gọi tập dữ liệu có kiểu (K,V) và (K,W), nó sẽ trả về 1 tập dữ liệu(K,seq(V),seq(W)). |
| cartesian(*otherDataset*) | Khi gọi 1 tập dữ liệu kiểu T và U, nó sẽ trả về tập dữ liệu mới (T,U). |
| keys() |  |
| keyBy() |  |
| values() |  |
| mapValue() |  |
| foldByKey() |  |

### Basic RDD Actions

Qua 1 phương thức actions thì sẽ cho phép trả về 1 giá trị cho driver program sau khi chạy tính toán trên tập dữ liệu.

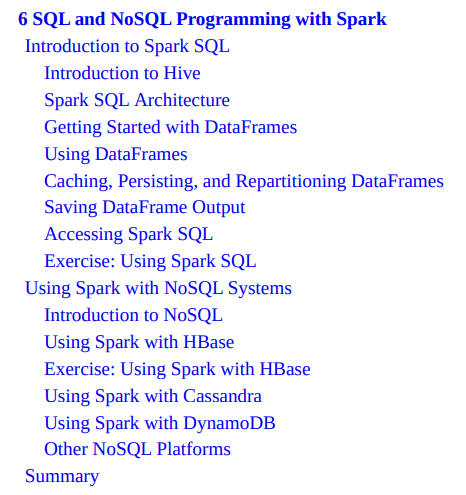
|  |  |
| --- | --- |
| Action | Ý nghĩa |
| reduce(*func*) | Tổng hợp các phần tử của tập dữ liệu sử dụng hàm *func* (có 2 đối và trả về 1 kết quả). |
| collect() | Trả về tất cả các phần tử của tập dữ liệu như 1 mảng ở driverProgram. Hàm này hữu ích sau khi lọc hoặc thao tác khác mà trả về tập dữ liệu con đủ nhỏ. |
| count() | Trả về số phần tử của tập dữ liệu |
| top() |  |
| first() | Trả về phần tử đầu tiên của tập dữ liệu( tương tự take(1)). |
| fold() | Tương tự reduce(), nhưng có thể hoạt động trên tập rỗng hoặc 1 pt duy nhất |
| take(*n*) | Trả về mảng gồm n phần tử đầu tiên của tập dữ liệu. |
| saveAsTextFile(*path*) | Ghi các phần tử của tập dữ liệu như 1 file text( hoặc tập file text)lên 1 thư mục trong hệ thống local, HDFS hoặc hệ thống hỗ trợ Hadoop bất kỳ. |
| countByKey() | Chỉ cho RDD có kiểu (K,V). Trả về 1 Map (K,Int). Int là chỉ số key. |
| foreach(*func*) | Chạy hàm *func*cho mỗi phần tử của tập dữ liệu. Điều này có tác dụng khi thực hiện cập nhật 1 biến accumulator hoặc tương tác với hệ thống lưu trữ ngoài. |

### Join Transformations

### Transformations on Sets/Numeric RDDs

## Summary

# **Chapter 6 – SQL and NoSQL Programming with Spark**

In This Chapter:

* Introduction to Hive and Spark SQL
* Introduction to the SparkSession object and DataFrame API
* Creating and accessing Spark DataFrames
* Using Spark SQL with external applications
* Introduction to NoSQL concepts and systems
* Using Spark with HBase, Cassandra, and DynamoDB

## Introduction to Spark SQL

Spark SQL là một trong những phần mở rộng phổ biến nhất của Spark. Spark SQL cho phép các truy vấn tương tác, hỗ trợ trí tuệ kinh doanh (BI) và các công cụ trực quan hóa và làm cho Spark có thể tiếp cận được với nhiều đối tượng phân tích rộng hơn. Trước khi tìm hiểu sâu về nội dung bên trong của Spark SQL, trước tiên tôi nên trình bày về bối cảnh và bối cảnh lịch sử đằng sau sáng kiến, bao gồm cả dự án Hive.

## 1. Mục tiêu

Apache SparkQuery là một mô-đun Spark để đơn giản hóa việc làm việc với dữ liệu có cấu trúc bằng cách sử dụng các tóm tắt DataFrame và Data set trong Python, Java và  [Scala](https://data-flair.training/blogs/why-you-should-learn-scala-introductory-tutorial/) . Những tóm tắt này là tập hợp dữ liệu phân tán được sắp xếp thành các cột được đặt tên. Nó cung cấp một kỹ thuật tối ưu hóa tốt. Sử dụng Spark SQL, chúng ta có thể truy vấn dữ liệu, cả từ bên trong chương trình Spark và từ các công cụ bên ngoài kết nối thông qua các trình kết nối cơ sở dữ liệu tiêu chuẩn (JDBC / ODBC) đến Spark SQL.

## 2. Hướng dẫn Apache Spark SQL

### 2.1. Giới thiệu Spark SQL

Apache Spark SQL là một mô-đun để xử lý dữ liệu có cấu trúc trong Spark. Sử dụng giao diện được cung cấp bởi Spark SQL, chúng tôi nhận được thêm thông tin về cấu trúc của dữ liệu và các tính toán được thực hiện. Với thông tin bổ sung này, người ta có thể đạt được tối ưu hóa thêm trong Apache Spark. Chúng tôi có thể tương tác với Spark SQL theo nhiều cách khác nhau như [DataFrame](https://data-flair.training/blogs/apache-spark-sql-dataframe-tutorial/) và [API bộ dữ liệu](https://data-flair.training/blogs/apache-spark-dataset-tutorial/) . Công cụ thực thi tương tự được sử dụng trong khi tính toán một kết quả, bất kể API / ngôn ngữ nào chúng tôi sử dụng để thể hiện tính toán. Do đó, người dùng có thể dễ dàng chuyển đổi qua lại giữa các API khác nhau.

Trong Apache Spark SQL, chúng ta có thể sử dụng dữ liệu có cấu trúc và bán cấu trúc theo bốn cách:

* Để đơn giản hóa việc làm việc với dữ liệu có cấu trúc, nó cung cấp các tóm tắt DataFrame trong Python, Java và Scala. DataFrame là một tập hợp dữ liệu phân tán được tổ chức thành các cột được đặt tên. Nó cung cấp một kỹ thuật tối ưu hóa tốt.
* Dữ liệu có thể được đọc và ghi trong nhiều định dạng có cấu trúc. Ví dụ: JSON,  [Hive](https://data-flair.training/blogs/apache-hive-tutorial-introductory-guide/)  Table và Parquet.
* Sử dụng SQL, chúng ta có thể truy vấn dữ liệu, cả từ bên trong chương trình Spark và từ các công cụ bên ngoài. Công cụ bên ngoài kết nối thông qua các trình kết nối cơ sở dữ liệu tiêu chuẩn (JDBC / ODBC) với Spark SQL.
* Cách tốt nhất để sử dụng Spark SQL là bên trong ứng dụng Spark. Điều này cho phép chúng tôi tải dữ liệu và truy vấn nó bằng SQL. Đồng thời, chúng ta cũng có thể kết hợp nó với mã chương trình thông thường của NX trong Python, Java hoặc Scala.

Khi SQL chạy từ ngôn ngữ lập trình khác, kết quả sẽ là Bộ dữ liệu / Khung dữ liệu. Sự tương tác với giao diện SQL được thực hiện bằng cách sử dụng dòng lệnh hoặc qua JDBC / ODBC.

### 2.2. Dữ liệu Spark Spark

Có một số [hạn chế với RDD.](https://data-flair.training/blogs/apache-spark-rdd-limitations/)Khi làm việc với dữ liệu có cấu trúc, không có công cụ tối ưu hóa tích hợp. Trên cơ sở các thuộc tính, nhà phát triển phải tối ưu hóa từng [RDD](https://data-flair.training/blogs/apache-spark-rdd-tutorial/) . Ngoài ra, không có quy định để xử lý dữ liệu có cấu trúc. DataFrame trong Spark SQL khắc phục những hạn chế này của RDD. Spark DataFrame là phiên bản Spark 1.3. Nó là một tập hợp phân phối dữ liệu được sắp xếp vào các cột được đặt tên. Khái niệm khôn ngoan, nó bằng với bảng trong cơ sở dữ liệu quan hệ hoặc khung dữ liệu trong [R](https://data-flair.training/blogs/r-programming-tutorial/) / Python. Chúng tôi có thể tạo DataFrame bằng cách sử dụng:

* Tệp dữ liệu có cấu trúc
* Bàn trong tổ ong
* Cơ sở dữ liệu bên ngoài
* Sử dụng RDD hiện có

### 2.3. Bộ dữ liệu Spark SQL

Spark Dataset là một giao diện được thêm vào trong phiên bản Spark 1.6. nó là một bộ sưu tập dữ liệu phân tán. Bộ dữ liệu cung cấp các [lợi ích của RDD](https://data-flair.training/blogs/apache-spark-rdd-features/) cùng với các lợi ích của công cụ thực thi được tối ưu hóa của Apache Spark SQL. Ở đây, bộ mã hóa là một khái niệm thực hiện chuyển đổi giữa các đối tượng JVM và các biểu diễn dạng bảng.

Một bộ dữ liệu có thể được tạo bằng các đối tượng JVM và sau đó, nó có thể được xử lý bằng các phép biến đổi chức năng (ánh xạ, bộ lọc, v.v.). API Dataset có thể truy cập inScala và Java. API Dataset không được Python hỗ trợ, nhưng do tính chất động của Python, nhiều lợi ích của API Dataset có sẵn. Trường hợp tương tự với R. Sử dụng Bộ dữ liệu của các hàng chúng tôi biểu thị DataFrame trong Scala và Java.

### 2.4. Tối ưu hóa chất xúc tác Spark

Trình tối ưu hóa được Spark SQL sử dụng là trình tối ưu hóa Catalyst. Nó tối ưu hóa tất cả các truy vấn được viết bằng Spark SQL và DataFrame DSL. Trình tối ưu hóa giúp chúng tôi chạy các truy vấn nhanh hơn nhiều so với đối tác RDD của chúng. Điều này làm tăng hiệu suất của hệ thống.

Spark Catalyst là một thư viện được xây dựng như một hệ thống dựa trên quy tắc. Và mỗi quy tắc tập trung vào việc tối ưu hóa cụ thể. Ví dụ: ConstantFoldingtrọng tâm của việc là loại bỏ một  biểu thức không đổi khỏi truy vấn.

### 2.5. Công dụng của Apache Spark SQL

* Nó thực thi các truy vấn SQL.
* Chúng tôi có thể đọc dữ liệu từ các bản cài đặt hiện có bằng SparkQuery.
* Khi chúng tôi chạy SQL trong một ngôn ngữ lập trình khác, chúng tôi sẽ nhận được kết quả dưới dạng Bộ dữ liệu / Khung dữ liệu.

### 2.6. Các hàm được xác định bởi Spark SQL

* Hàm dựng sẵn: Cung cấp hàm dựng sẵn để xử lý giá trị cột. Chúng ta có thể truy cập hàm inbuilt bằng cách sử dụng lệnh sau:
  + Import org.apache.spark.sql.functions
* Hàm do người dùng xác định (UDF): UDF cho phép bạn tạo các hàm định nghĩa người dùng dựa trên các hàm do người dùng xác định trong Scala.
* Hàm tổng hợp: Chúng hoạt động trên một nhóm hàng và tính một giá trị trả về duy nhất cho mỗi nhóm.
* Windowed Aggregates (Windows): Chúng hoạt động trên một nhóm các hàng và tính toán một giá trị trả về duy nhất cho mỗi hàng trong một nhóm.

### 2.7. Ưu điểm của Spark SQL

Trong phần này, chúng ta sẽ thảo luận về các lợi thế khác nhau của Apache Spark SQL.

#### a. Tích hợp

Apache Spark SQL trộn các truy vấn SQL với các chương trình Spark. Với sự trợ giúp của Spark SQL, chúng tôi có thể truy vấn dữ liệu có cấu trúc dưới dạng tập dữ liệu phân tán (RDD). Chúng ta có thể chạy các truy vấn SQL cùng với các thuật toán phân tích phức tạp bằng cách sử dụng thuộc tính tích hợp chặt chẽ của Spark SQL.

#### b. Truy cập dữ liệu hợp nhất

Sử dụng Spark SQL, chúng tôi có thể tải và truy vấn dữ liệu từ các nguồn khác nhau. Các Schema-RDD cho phép các giao diện đơn hoạt động hiệu quả với dữ liệu có cấu trúc. Ví dụ: các bảng Apache Hive, các tệp sàn gỗ và các tệp JSON.

#### c. Khả năng tương thích cao

Trong Apache Spark SQL, chúng ta có thể chạy các truy vấn Hive chưa sửa đổi trên các kho hiện có. Nó cho phép tương thích hoàn toàn với dữ liệu Hive, truy vấn và UDF hiện có, bằng cách sử dụng giao diện Hive và MetaStore.

#### d. Kết nối tiêu chuẩn

Nó có thể kết nối thông qua JDBC hoặc ODBC. Nó bao gồm chế độ máy chủ với kết nối JDBC và ODBC tiêu chuẩn công nghiệp.

#### e. Khả năng mở rộng

Để hỗ trợ khả năng chịu lỗi truy vấn giữa và các công việc lớn, cần tận dụng mô hình RDD. Nó sử dụng cùng một công cụ cho các truy vấn tương tác và dài.

#### f. Tối ưu hóa hiệu suất

Công cụ tối ưu hóa truy vấn trong Spark SQL chuyển đổi từng truy vấn SQL thành một kế hoạch logic. Hơn nữa, nó chuyển đổi sang nhiều kế hoạch thực hiện vật lý. Trong toàn bộ kế hoạch, nó chọn phương án vật lý tối ưu nhất để thực hiện.

#### g. Để xử lý hàng loạt các bảng Hive

Chúng ta có thể sử dụng Spark SQL để xử lý hàng loạt các bảng Hive nhanh chóng.

### 2.8. Nhược điểm của Spark SQL

Ngoài những tính năng này, còn có một số nhược điểm của Spark SQL. Một số trong số họ được liệt kê dưới đây.

#### a. Loại liên minh không hỗ trợ

Sử dụng Spark SQL, chúng tôi không thể tạo hoặc đọc bảng chứa các trường kết hợp.

#### b. Không có lỗi cho quá khổ của loại Varchar

Ngay cả khi giá trị được chèn vượt quá giới hạn kích thước, sẽ không có lỗi xảy ra. Dữ liệu tương tự sẽ bị cắt bớt nếu đọc từ Hive nhưng không được đọc từ Spark. SparkQuery sẽ coi varchar là một chuỗi, có nghĩa là không có giới hạn kích thước.

#### c. Không hỗ trợ cho bảng giao dịch

Giao dịch Hive không được Spark SQL hỗ trợ.

#### d. Loại Char không hỗ trợ

Loại Char (chuỗi độ dài cố định) không được hỗ trợ. Giống như liên minh, chúng ta không thể đọc hoặc tạo bảng với các trường như vậy.

#### e. Không hỗ trợ cho Dấu thời gian trong Bảng Avro.

## 3. Kết luận

Để kết luận với Spark SQL, đây là một mô-đun của Apache Spark phân tích dữ liệu có cấu trúc. Nó cung cấp khả năng mở rộng và đảm bảo khả năng tương thích cao của hệ thống. Nó có kết nối tiêu chuẩn thông qua JDBC hoặc ODBC. Do đó, nó cung cấp cách tự nhiên nhất để thể hiện dữ liệu có cấu trúc.

### Introduction to Hive

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.

Ban đầu Hive được phát triển bởi Facebook, sau đó Quỹ Phần mềm Apache đã lấy và phát triển nó thành một nguồn mở dưới tên Apache Hive. Nó được sử dụng bởi các công ty khác nhau. Ví dụ: Amazon sử dụng nó trong Amazon Elastic MapReduce.

Hive không phải là:

* Một CSDL quan hệ
* Một thiết kế để xử lý giao dịch Online (OnLine Transaction Processing - OLTP)
* Một ngôn ngữ cho các truy vấn thời gian thực và cập nhật cấp hàng

Đặc trưng của Hive

* Nó lưu trữ lược đồ trong cơ sở dữ liệu và xử lý dữ liệu vào HDFS.
* Nó được thiết kế cho OLAP.
* Nó cung cấp ngôn ngữ kiểu SQL để truy vấn được gọi là HiveQL hoặc HQL.
* Nó là quen thuộc, nhanh chóng, có khả năng mở rộng.

### Spark SQL Architecture

Kiến trúc Spark SQL: Kiến trúc này chứa ba lớp cụ thể là Language API, Schema RDD và Data Sources.

* Language API − Spark tương thích với các ngôn ngữ khác nhau và Spark SQL. Nó cũng được hỗ trợ bởi các ngôn ngữ này - API (python, scala, java, HiveQL).
* Schema RDD − Spark Core được thiết kế với cấu trúc dữ liệu đặc biệt được gọi là RDD. Nói chung, Spark SQL hoạt động trên sơ đồ, bảng và bản ghi. Do đó, chúng ta có thể sử dụng Schema RDD làm bảng tạm thời. Chúng ta có thể gọi Schema RDD này là Data Frame.
* Data Sources − Thông thường nguồn dữ liệu cho lõi thu nhỏ là tệp văn bản, tệp Avro, v.v. Tuy nhiên, nguồn dữ liệu cho Spark SQL là khác nhau. Đó là tệp Parquet, tài liệu JSON, bảng HIVE và cơ sở dữ liệu Cassandra.

### Getting Started with DataFrames

Spark SQL DataFrames là tập hợp các bản ghi được phân phối, tất cả đều giống nhaulược đồ được xác định, tương tự về mặt khái niệm với một bảng được phân đoạn từ một quan hệcơ sở dữ liệu.

DataFrames là một trừu tượng cho các RDD Spark. Tuy nhiên, không giống như nguyên thủy RDD, DataFrames theo dõi lược đồ của chúng và cung cấp hỗ trợ gốc cho nhiềucác hàm SQL phổ biến và các toán tử quan hệ. DataFrames, như RDD, làđược đánh giá là DAG, sử dụng đánh giá lười biếng và cung cấp lineage and fault tolerance. Cũng giống như RDD, DataFrames hỗ trợ bộ nhớ đệm và sử dụng liên tục các phương pháp tương tự.

DataFrames có thể được tạo theo nhiều cách khác nhau, bao gồm:

* RDD hiện có
* Tệp JSON
* Tệp văn bản, tệp Parquet hoặc tệp ORC
* Một bảng trong Hive
* Một bảng trong cơ sở dữ liệu bên ngoài
* Một bảng tạm thời trong Spark

#### Creating a DataFrame from an Existing RDD

* createDataFrame()

#### Creating a DataFrame from a Hive Table

* sql()
* table()

#### Creating DataFrames from JSON Objects

* read.json()

#### Creating DataFrames from Flat Files

* text()
* parquet()

#### Converting DataFrames to RDDs

* df.rdd

### Using DataFrames

* columns()
* dtypes()
* show()
* select()
* drop(col)
* filter()
* distinct()
* join()
* orderBy()
* groupBy()

### Caching, Persisting, and Repartitioning DataFrames

### Saving DataFrame Output

### Accessing Spark SQL

### Exercise: Using Spark SQL

Bài tập này chỉ ra cách khởi động máy chủ Spark SQL Thrift và sử dụng

tiện ích máy khách beeline để kết nối với máy chủ. Bạn sẽ tạo bảng Hive

dựa trên dữ liệu mẫu và sử dụng beeline và Thrift để chạy truy vấn SQL

dữ liệu, được thực thi bởi Spark SQL. Bạn sẽ sử dụng tập dữ liệu chia sẻ xe đạp được sử dụng cho

các bài tập trong chương trước. Làm theo các bước sau:

1. Khởi động máy chủ JDBC / ODBC Thrift:

Bấm vào đây để xem hình ảnh mã

$ sudo $ SPARK\_HOME / sbin / start-thriftserver.sh \

- quản trị viên địa phương \

--hiveconf hive.server2.thrift.port = 10001 \

--hiveconf hive.server2.thrift.bind.host = 10001

Thay vào đó, bạn có thể khởi động máy chủ ở chế độ YARN bằng cách sử dụng --master

sợi-cluster nếu bạn có sẵn một cụm YARN cho mình.

2. Mở phiên beeline:

$ SPARK\_HOME / bin / beeline

3. Tại lời nhắc beeline>, hãy tạo kết nối với máy chủ Thrift của bạn:

Bấm vào đây để xem hình ảnh mã

beeline>! connect jdbc: hive2: // localhost: 10001

Nhập tên người dùng cho jdbc: hive2: // localhost: 10001: hadoop

Nhập mật khẩu cho jdbc: hive2: // localhost: 10001: \*\*\*\*\*\*\*\*\*

Bạn được nhắc nhập tên người dùng và mật khẩu, như được hiển thị ở trên. Các

tên người dùng được cung cấp phải tồn tại trên máy chủ Thrift và có

quyền trên hệ thống tệp.

4. Sau khi bạn kết nối với máy chủ, hãy tạo bảng chuyến đi từ chia sẻ xe đạp

demo bằng cách nhập lệnh HiveQL DDL sau:

Bấm vào đây để xem hình ảnh mã

TẠO các chuyến đi BẢNG NGOÀI (

Int TripID,

|  |
| --- |
| **Trang 47** |

Thời lượng int,

Chuỗi ngày bắt đầu,

Chuỗi StartStation,

StartTerminal int,

Chuỗi ngày kết thúc,

Chuỗi EndStation,

Int EndTerminal,

BikeNo int,

Chuỗi SubscriberType, chuỗi ZipCode)

ROW ĐỊNH DẠNG BỊ XÓA

CÁC LĨNH VỰC ĐƯỢC CHẤM DỨT BỞI ','

Tệp LOCATION ': /// opt / spark / data / bike-share / trip /';

5. Thực thi truy vấn SQL sau đối với bảng bạn vừa tạo:

Bấm vào đây để xem hình ảnh mã

CHỌN StartTerminal, StartStation, COUNT (1) AS đếm

TỪ chuyến đi

NHÓM THEO StartTerminal, StartStation

ĐẶT HÀNG THEO số lượng DESC

GIỚI HẠN 10;

6. Xem giao diện người dùng web ứng dụng Spark của bạn để xác nhận rằng truy vấn của bạn đã được thực thi

sử dụng Spark SQL. Nhớ lại rằng điều này có thể truy cập được bằng cách sử dụng cổng 4040 của

localhost nếu bạn đang chạy Spark cục bộ hoặc máy chủ chính của ứng dụng nếu

bạn đang sử dụng YARN (có thể truy cập từ Giao diện người dùng Trình quản lý tài nguyên). Nhân vật

6.10 cũng hiển thị tab SQL trong giao diện người dùng ứng dụng.

## Using Spark with NoSQL Systems

### Introduction to NoSQL

Đây là thuật ngữ chung cho các hệ CSDL không sử dụng mô hình dữ liệu quan hệ. NoSQL đặc biệt nhấn mạnh đến mô hình lưu trữ cặp giá trị - khóa và hệ thống lưu trữ phân tán.

Một số đặc điểm:

* Phi quan hệ (hay không ràng buộc): relational - ràng buộc - thuật ngữ sử dụng đến các mối quan hệ giữa các bảng trong cơ sở dữ liệu quan hệ (RDBM) sử dụng mô hình gồm 2 loại khóa: khóa chính và khóa phụ (primary key + foreign key) để ràng buộc dữ liệu nhằm thể hiện tính nhất quán dữ liệu từ các bảng khác nhau. Non-relational là khái niệm không sử dụng các ràng buộc dữ liệu cho nhất quán dữ liệu.
* Lưu trữ phân tán: mô hình lưu trữ phân tán các tập tin hoặc dữ liệu ra nhiều máy khác nhau trong mạng LAN hoặc Internet dưới sự kiểm soát của phần mềm.
* Nhất quán cuối: tính nhất quán của dữ liệu không cần phải đảm bảo ngay tức khắc sau mỗi phép ghi. Một hệ thống phân tán chấp nhận những ảnh hưởng theo phương thức lan truyền và sau một khoảng thời gian (không phải ngay tức khắc), thay đổi sẽ đi đến mọi điểm trong hệ thống để cuối cùng dữ liệu trên hệ thống sẽ trở lại trạng thái nhất quán.
* Triển khai đơn giản, dễ nâng cấp và mở rộng.
* Mô hình dữ liệu và truy vấn linh hoạt. …

**1. Document database** (ví dụ: CouchDB, MongoDB): Dữ liệu được thêm vào lưu trữ dưới dạng cấu trúc JSON tự do hoặc “tài liệu”, trong đó dữ liệu có thể là bất kỳ kiểu nào, từ số nguyên đến chuỗi hay đến các văn bản tự do.

* *Ưu điểm*: Dùng khi dữ liệu nguồn không được mô tả đầy đủ.
* *Nhược điểm*: Hiệu năng truy vấn, Không có cú pháp chuẩn cho câu truy vấn dữ liệu.

**2. Key-value stores** (ví dụ: Redis, Riak): Các giá trị dạng tự do, từ số nguyên hoặc chuỗi đơn giản đến các tài liệu JSON phức tạp, được truy cập trong cơ sở dữ liệu bằng các khóa.

* *Ưu điểm*: Tìm kiếm rất nhanh.
* *Nhược điểm*: Lưu dữ liệu không theo khuôn dạng (schema) nhất định.

**3. Wide column stores** (ví dụ: HBase, Cassandra): Dữ liệu được lưu trữ trong các cột thay vì các hàng như trong một hệ thống SQL thông thường. Bất kỳ số lượng cột nào (và do đó nhiều loại dữ liệu khác nhau) có thể được nhóm hoặc tổng hợp khi cần cho truy vấn hoặc chế độ xem dữ liệu.

* *Ưu điểm*: Tìm kiếm nhanh, Phân tán dữ liệu tốt.
* *Nhược điểm*: Hỗ trợ được với rất ít phần mềm.

**4. Graph database** (ví dụ: Neo4j): Dữ liệu được biểu diễn dưới dạng mạng hoặc đồ thị của các thực thể và các mối quan hệ của thực thể đó, với mỗi node trong biểu đồ là một khối dữ liệu ở dạng tự do.

* *Ưu điểm:* Ứng dụng các thuật toán trên đồ thị như Đường đi ngắn nhất, liên thông,…
* *Nhược điểm:* Phải duyệt nội bộ đồ thị, để trả lời lại các truy vấn. Không dễ để phân tán.

### Using Spark with Hbase(column stores)

Hbase là hệ quản trị cơ sở dữ liệu dựa trên Hadoop, đây là mã nguồn mở nằm trong dự án của Apache, phát triển và mở rộng từ dự án lưu trữ Bigdata của google. (được xây dựng dựa trên Google Big Table). Hbase được viết bằng ngôn ngữ Java, có thể lưu trữ dữ liệu cực lớn từ terabytes đến petabytes.

Hbase thực chất là một NoSQL điển hình nên vì thế các table của Hbase không có một schemas cố định nào và cũng không có mối quan hệ giữa các bảng. Hiện nay, có rất nhiều công ty và tập đoàn công nghệ lớn trên thế giới sử dụng Hbase, có thể kể đến: Facebook, Twitter, Yahoo, Adobe….

### Exercise: Using Spark with HBase

### Using Spark with Cassandra

[Cassandra](http://cassandra.apache.org/) là NoSQL, được phát triển bởi Facebook vào năm 2007. Sau đó nó được tặng cho quỹ Apache vào 2/2010 và nâng cấp lên thành dự án hàng đầu của 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.

### Using Spark with DynamoDB

Amazon DynamoDB là cơ sở dữ liệu về khóa-giá trị và tài liệu, với hiệu năng hoạt động chỉ trong vài mili giây ở mọi quy mô. Đây là một cơ sở dữ liệu bền vững, đa hoạt tính, đa khu vực, được quản lý toàn phần, có tích hợp tính năng bảo mật, sao lưu, khôi phục và lưu vào bộ nhớ đệm cho các ứng dụng trên quy mô internet. DynamoDB có thể xử lý hơn 10 nghìn tỷ yêu cầu/ngày và hỗ trợ tối đa hơn 20 triệu yêu cầu/giây.

Nhiều doanh nghiệp phát triển nhanh nhất thế giới như Lyft, Airbnb, Redfin, Samsung, Toyota và Capital One dựa vào quy mô và hiệu năng của DynamoDB để hỗ trợ khối lượng công việc then chốt của họ.

### Other NoSQL Platforms