2021

Nội dung

- Giới thiệu
- Resilient Distributed Datasets (RDDs)
- Cài đặt Spark
- Lập trình Spark
- Triển khai ứng dụng

Giới thiệu

- Apache Spark là nền tảng xử lý dữ liệu lớn với tốc độ tính toán rất nhanh.
- Spark được xây dựng trên nền Hadoop MapReduce, đồng thời kế thừa một số tính năng của MapReduce như xử lý luồng (stream processing).
- Thế mạnh của Hadoop bao gồm mô hình lập trình đơn giản (MapReduce), dễ mở rộng, khả năng chịu lỗi cao, chi phí hợp lý, ngoại trừ tốc độ.
- Spark được giới thiệu để tăng tốc độ xử lý trên nền tảng Hadoop. Tuy nhiên Spark không phải là phiên bản cải tiến của Hadoop hay phụ thuộc vào Hadoop; có thể cài đặt Spark độc lập do nó có hệ thống quản lý dữ liệu riêng.
- Spark dùng Hadoop cho hai việc chính: **lưu trữ** và **xử lý**.

Đặc điểm chính của Spark

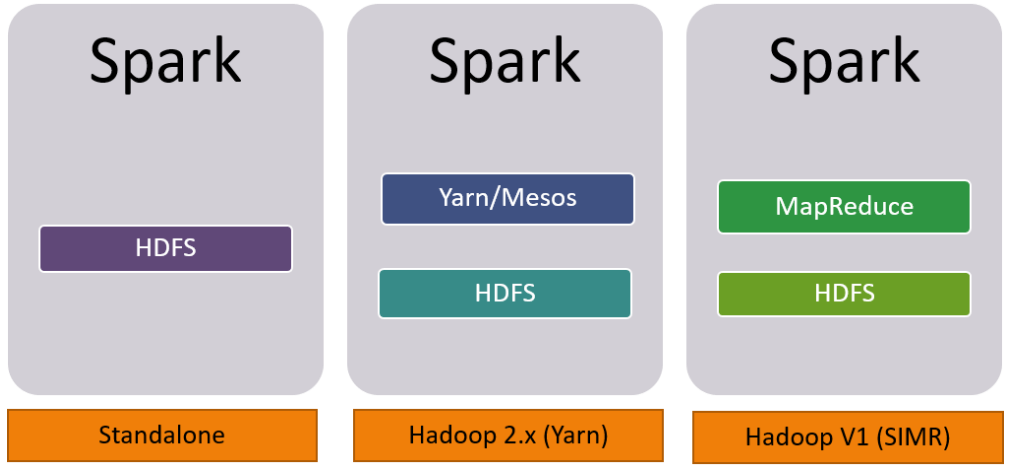
- Lịch sử phát triển
  - 2009: Spark là dự án con của Hadoop phát triển tại UC Berkeley's AMPLab bởi Matei Zaharia.
  - 2010: Trở thành nguồn mở.
  - 2013: Được tặng cho Apache Software Foundation.
  - 2014: Apache Spark trở thành dự án cấp cao nhất tại Apache Software Foundation.
- Đặc điểm chính
  - Tốc độ (speed): Spark thực hiện tính toán bên trong bộ nhớ (in-memory cluster computing); tính năng này giúp tăng tốc độ tính toán lên nhiều lần so với xử lý trên bộ nhớ ngoài.
  - Spark hỗ trợ đa dạng ứng dụng: Ứng dụng theo lô, các giải thuật lặp lại, streaming.

Đặc điểm chính của Spark (cont.)

- Đặc điểm chính (cont.)
  - Dễ sử dụng & Hỗ trợ đa ngôn ngữ lập trình: Java, Scala, Python, R.
  - Nhiều mô hình xử lý cao cấp: Ngoài MapReduce, Spark còn hỗ trợ truy vấn SQL, streaming data, học máy (machine learning), và các giải thuật xử lý đồ thị (graphs).
- Spark chạy được trên nhiều nền tảng: Hadoop, Apache Mesos, Kubernetes, standalone, hoặc trên nền tảng điện toán đám mây (cloud).
- Spark hỗ trợ đa dạng nguồn quản lý dữ liệu: HDFS, Alluxio, Apache Cassandra, Apache HBase, Apache Hive ...

Spark trên Hadoop

- Có 3 cách triển khai ứng dụng Spark trên Hadoop:
  - Standalone
  - Spark on Yarn/Mesos
  - Spark in MapReduce

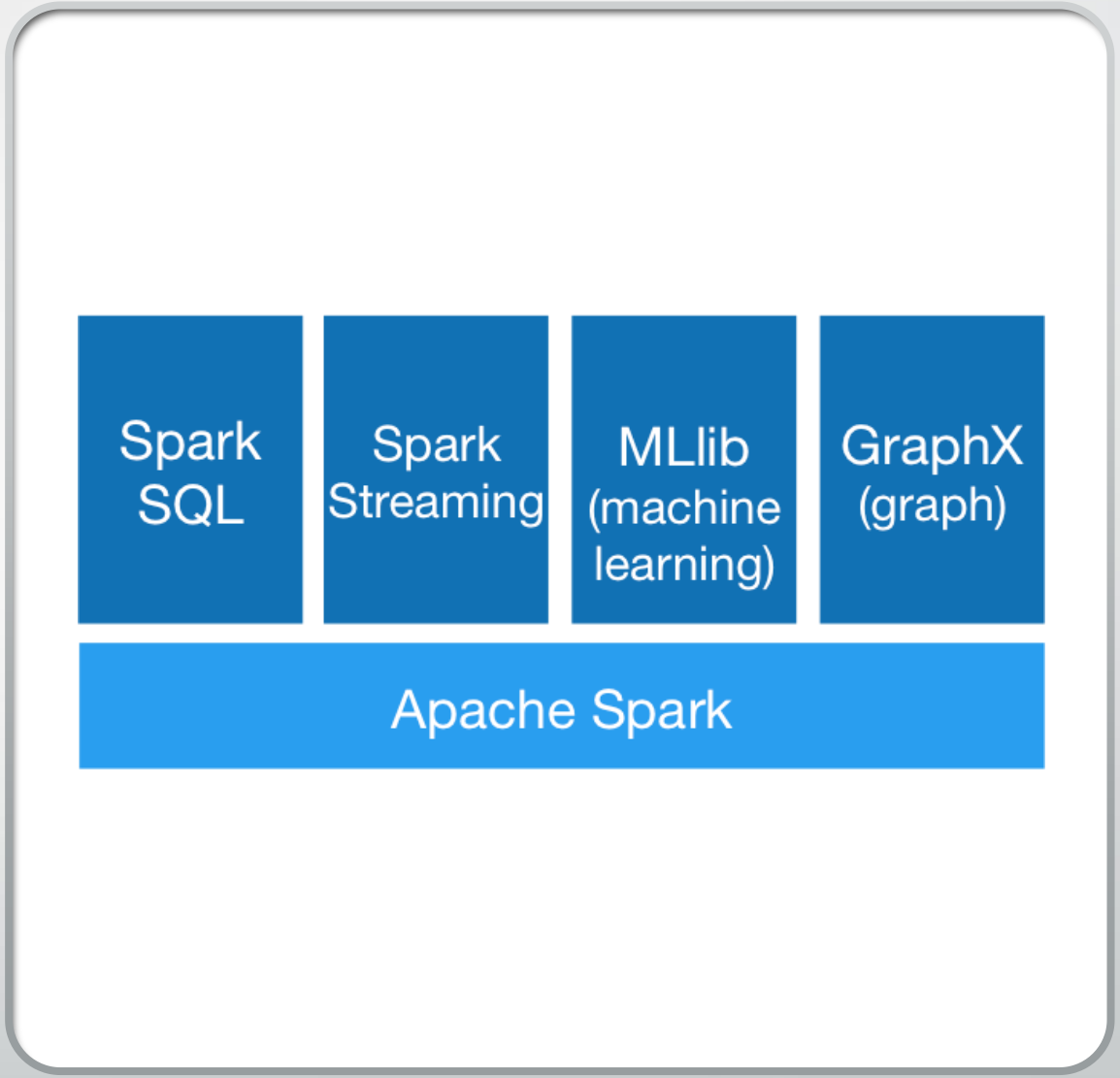


Spark trên Hadoop

- Standalone
  - Ứng dụng Spark chạy trên HDFS(Hadoop Distributed File System).
  - Ứng dụng Spark và MapReduce sẽ chạy cùng nhau.
- Spark on Yarn/Mesos
  - Ứng dụng Spark chạy trên nền tảng Yarn mà không cần cài đặt trước hay cấp quyền quản trị cao nhất.
  - Cho phép tích hợp Spark vào hệ sinh thái Hadoop.
- Spark in MapReduce (SIMR)
  - Spark trên MapReduce dùng để chạy ứng dụng Spark
  - Với SIMR người dùng có thể khởi động và sử dụng Spark mà không cần cấp quyền quản trị.

Các thành phần Spark

- Kết hợp SQL, streaming, và các tác vụ phân tích phức hợp.
- Spark cung cấp nhiều thư viện: SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming.
- Có thể kết hợp các thư viện này trong cùng một ứng dụng.



Các thành phần Spark (cont.)

- Apache Spark Core
  - Là hệ thống cốt lõi thực thi tổng quát, làm cơ sở xây dựng các chức năng khác.
  - Cung cấp tính năng tính toán trong bộ nhớ (In-Memory computing) và tham chiếu dữ liệu trên bộ nhớ ngoài.
- Spark SQL
  - Là thành phần được xây dựng trên nền Spark Core, cung cấp tính năng trừu tượng hóa dữ liệu (data abstraction) gọi là SchemaRDD, cho phép xử lý dữ liệu có cấu trúc và bán cấu trúc.

Các thành phần Spark (cont.)

- Spark Streaming
  - Spark Streaming khai thác tính năng lập lịch nhanh của Spark Core để thực hiện các tác vụ xử lý luồng (streaming). Dữ liệu được chia thành các gói nhỏ và thực hiên chuyển đổi RDD trên chúng.
- MLlib (Machine Learning Library)
  - Là nền tảng học máy phân tán trên Spark.
- GraphX
  - Là nền tảng xử lý đồ thị phân tán (distributed graph-processing) trên Spark.

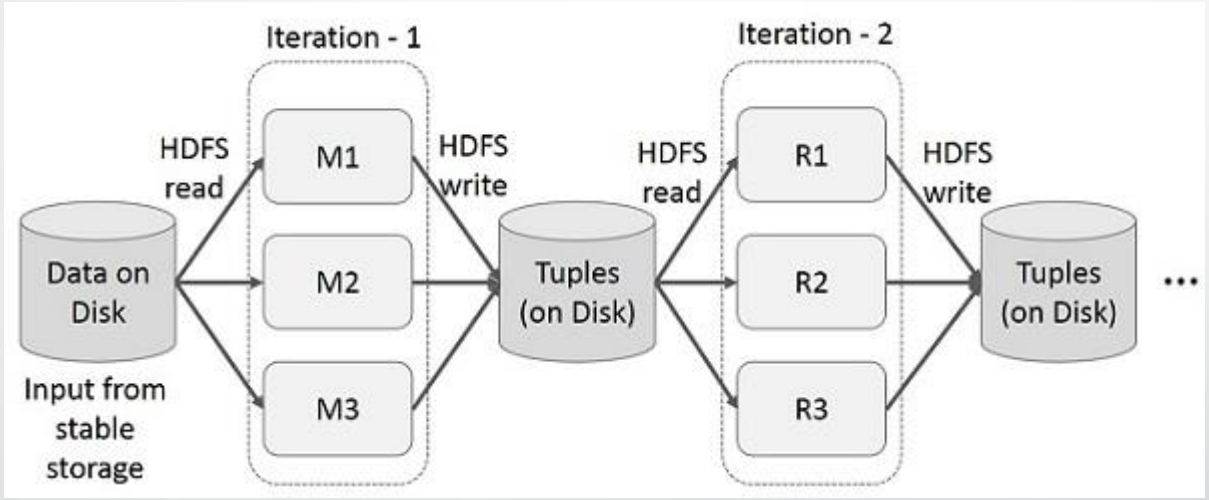
Resilient Distributed Datasets (RDDs)

- RDD là cấu trúc dữ liệu cơ bản của Spark.
- Một RDD là một tập các mẫu tin chỉ đọc, được phân chia logic thành các phân vùng. Mỗi phân vùng có thể được xử lý trên một node của cluster.
- RDD có thể chứa các cấu trúc dữ liệu của Python, Java hay Scala gồm cả các lớp tự định nghĩa.
- Có 2 cách tạo RDD
  - Song song (Parallelizing)
  - Tham chiếu (Referencing) một cơ sở dữ liệu ở bộ nhớ ngoài: HDFS, HBase

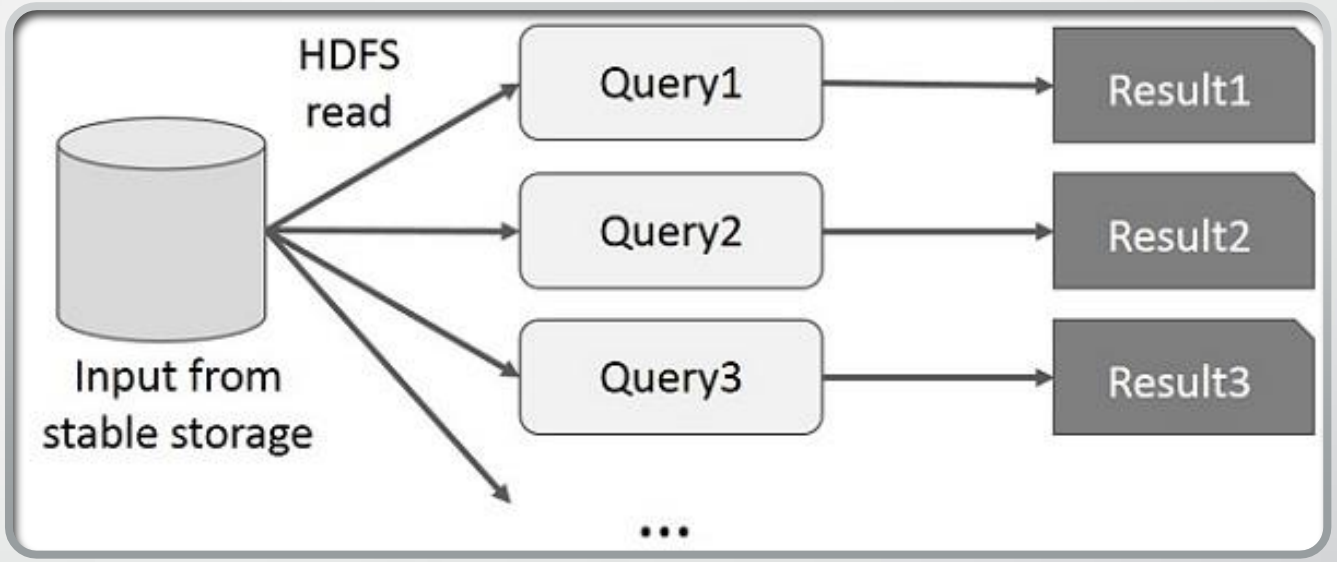
Chia sẻ dữ liệu trong MapReduce

- MapReduce đang được sử dụng rộng rãi trong xử lý dữ liệu lớn.
- Tuy nhiên, trên MapReduce, cách duy nhất để dùng lại dữ liệu giữa các tác vụ tính toán (vd, giữa các ứng dụng MapReduce) là ghi dữ liệu đó lên bộ nhớ ngoài (HDFS) .
- Hầu hết ứng dụng Hadoop sử dụng hơn 90% thời gian xử lý tác vụ đọc/ghi dữ liệu!
- Các ứng dụng có vòng lặp lại (iterative) và tương tác (interactive) đòi hỏi tốc độ chia sẻ dữ liệu nhanh hơn giữa các tác vụ song song. Chẳng hạn, huấn luyện một mô hình học máy (machine learning) đòi hỏi lặp lại hàng trăm epochs -> MapReduce không hiệu quả.

Iterative Operations on MapReduce

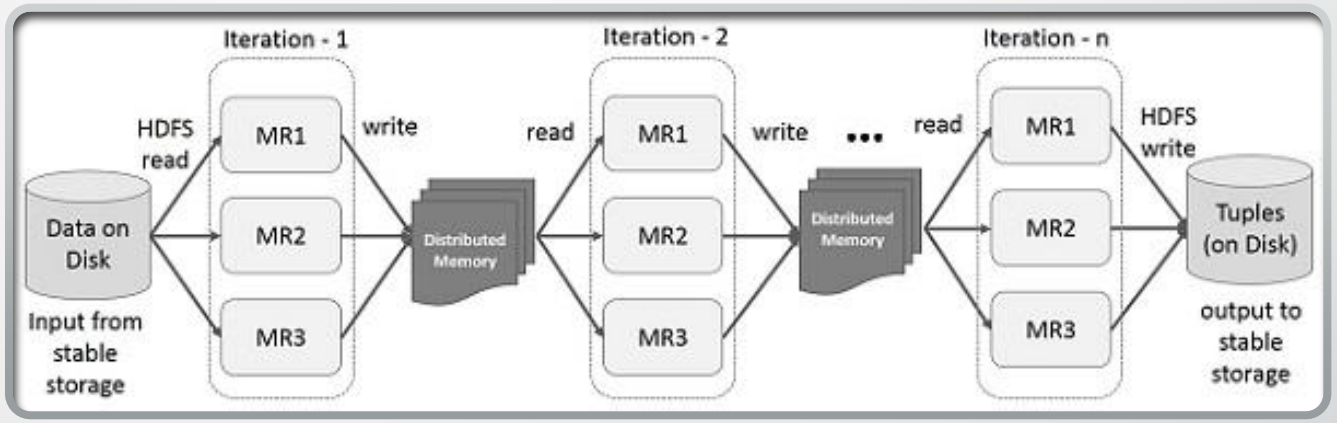


Interactive Operations on MapReduce



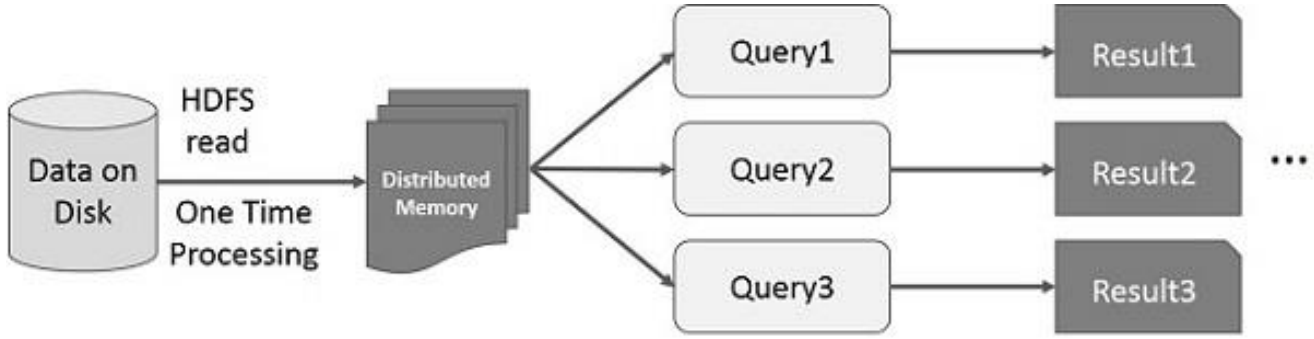
Iterative Operations on Spark RDD

- Spark lưu kết quả trung gian trong bộ nhớ trong (RAM)
- -> Nhanh hơn rất nhiều so với MapReduce.



Interactive Operations on Spark RDD

- Các truy vấn thực hiện lặp lại trên cùng dữ liệu
- Dữ liệu được lưu trong bộ nhớ RAM -> tốc độ nhanh hơn



PySpark Tutorial

- PySpark là phiên bản Apache Spark dùng cho Python.
- PySpark được tích hợp trong gói Apache Spark.
- PySpark cung cấp **PySpark Shell** liên kết Python API với Spark Core.
- Why PySpark?
  - Lĩnh vực khoa học dữ liệu ngày nay chủ yếu dùng Python
  - Python + Spark -> PySpark kết hợp sức mạnh của Python và Spark

Cài đặt PySpark

- Cài đặt PySpark với PyPI

  pip install pyspark

  pip3 [--user] install pyspark

- Cài đặt PySpark với Hadoop

  PYSPARK_HADOOP_VERSION = 3.2 pip install pyspark

- Cài PySpark dùng Conda

  conda install pyspark

- Cài đặt từ nguồn

Cài đặt PySpark

```
hung@hung-VirtualBox:~$ pip3 install --user pyspark
Collecting pyspark
  Downloading https://files.pythonhosted.org/packages/45/b0/9d6860891ab14a39d4bddf80ba26ce51c2f9dc4805e5c6978ac0472c120a/pyspark-3.1.1.tar.gz (212.3MB)
    100% |████████████████████████████████| 212.3MB 5.7kB/s
Collecting py4j==0.10.9 (from pyspark)
  Using cached https://files.pythonhosted.org/packages/9e/b6/6a4fb90cd235dc8e265a6a2067f2a2c99f0d91787f06aca4bcf7c23f3f80/py4j-0.10.9-py2.py3-none-any.whl
Building wheels for collected packages: pyspark
  Running setup.py bdist_wheel for pyspark ... done
  Stored in directory: /home/hung/.cache/pip/wheels/0b/90/c0/01de724414ef122bd05f056541fb6a0ecf47c7ca655f8b3c0f
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9 pyspark-3.1.1
hung@hung-VirtualBox:~$ pyspark
Python 3.6.9 (default, Jan 26 2021, 15:33:00)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
21/04/08 07:34:04 WARN Utils: Your hostname, hung-VirtualBox resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
21/04/08 07:34:04 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/04/08 07:34:05 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.1.1
      /_/

Using Python version 3.6.9 (default, Jan 26 2021 15:33:00)
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1617842047994).
SparkSession available as 'spark'.
>>>
```

Spark's Python Shells

- Spark cung cấp tính năng tương tác (interactive shells), cho phép thực hiện xử lý dữ liệu trên môi trường dòng lệnh khi cần, tương tự Python, R
- Điểm khác biệt của Spark shell là có khả năng xử lý với dữ liệu phân tán, trên đĩa từ hoặc trong bộ nhớ, tự động
- Khởi động PySpark – phiên bản Spark dành cho Python:

$SPARK_HOME/bin pyspark

```
hung@hung-VirtualBox:~$ pyspark
Python 3.6.9 (default, Jan 26 2021, 15:33:00)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
21/04/13 08:21:07 WARN Utils: Your hostname, hung-VirtualBox resolves to a loopback address:
21/04/13 08:21:07 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
21/04/13 08:21:08 WARN NativeCodeLoader: Unable to load native-hadoop library for your platf
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.1.1
      /_/

Using Python version 3.6.9 (default, Jan 26 2021 15:33:00)
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-1618276871146).
SparkSession available as 'spark'.
>>>
```

- VD: Đếm từ với PySpark shells:

```
>>> lines=sc.textFile("/home/hung/Downloads/gutenberg")
>>> lines.count()
78753
```

Spark: Các khái niệm căn bản

- Mọi ứng dụng Spark có một trình điều khiển (*driver program)* có chức năng nạp các tác vụ song song lên một cluster.
- Trình điều khiển chứa hàm main của chương trình ứng dụng, tạo dữ liệu phân tán trên cluster, sau đó thực hiện các tác vụ xử lý trên dữ liệu này.
- Trình điều khiển truy cập Spark thông qua đối tượng lớp SparkContext, tạo kết nối tới cluster.
- Trong ví dụ trước (word count): Spark shells đóng vai trò trình điều khiển, người dùng nhập các lệnh xử lý trên shells; Đối tượng SparkContext được tạo ra tự động với tên biến sc.

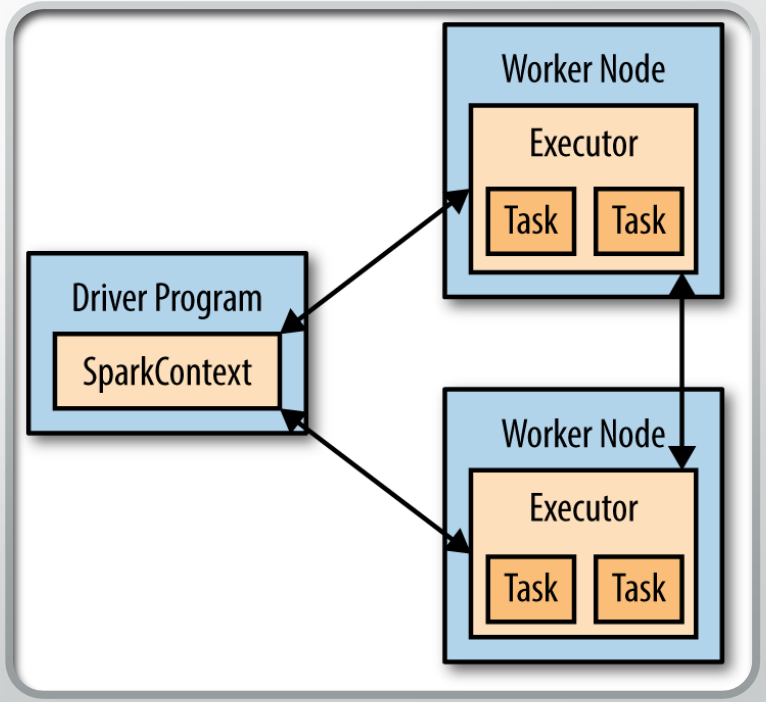
Spark Context

- Sau khi khởi tạo Spark Context, có thể tạo RDDs trên cluster. Ví dụ:

sc.textFile(<path to text file or directory>)

- Để thực thi các tác vụ, trình điều khiển gọi một số máy trạm, gọi là *executors*.
- Spark cung cấp các API xử lý trên cluster.
- Ví dụ: PySpark shell lọc ra các dòng có chứa từ "error" từ file log:

```
>>> lines = sc.textFile("/home/hung/Downloads/apache_logs")
>>> error_lines = lines.filter(lambda line: "error" in line)
>>> error_lines.first()
```



Gọi hàm trong Spark

- Có thể định nghĩa hàm và gọi nó từ Spark.
- Ví dụ, đoạn code PySpark kiểm tra một dòng văn bản có chứa từ "error" hay không:

```
def isError(line):
    return "error" in line
pythonLines = lines.filter(isError)
```

Chạy ứng dụng từ Spark

- Spark có thể kết nối và thực thi các ứng dụng độc lập viết bằng Java, Scala hay Python
- Khác với chạy Spark từ shell, để chạy các ứng dụng độc lập cần khởi tạo SparkContext.
- Cú pháp chạy chương trình Python từ Spark:

$spark-submit <my_script.py>

Khởi tạo SparkContext

- Sau khi kết nối Spark với chương trình (độc lập), cần nạp các gói Spark cần thiết,tạo đối tượng SparkConf để thiết lập cấu hình ứng dụng và khởi tạo SparkContext
- *Ví dụ: Khởi tạo Spark trong Python*

```
from pyspark import SparkConf, SparkContext

conf = SparkConf().setMaster("local").setAppName("Word Count")

sc = SparkContext(conf = conf)
```

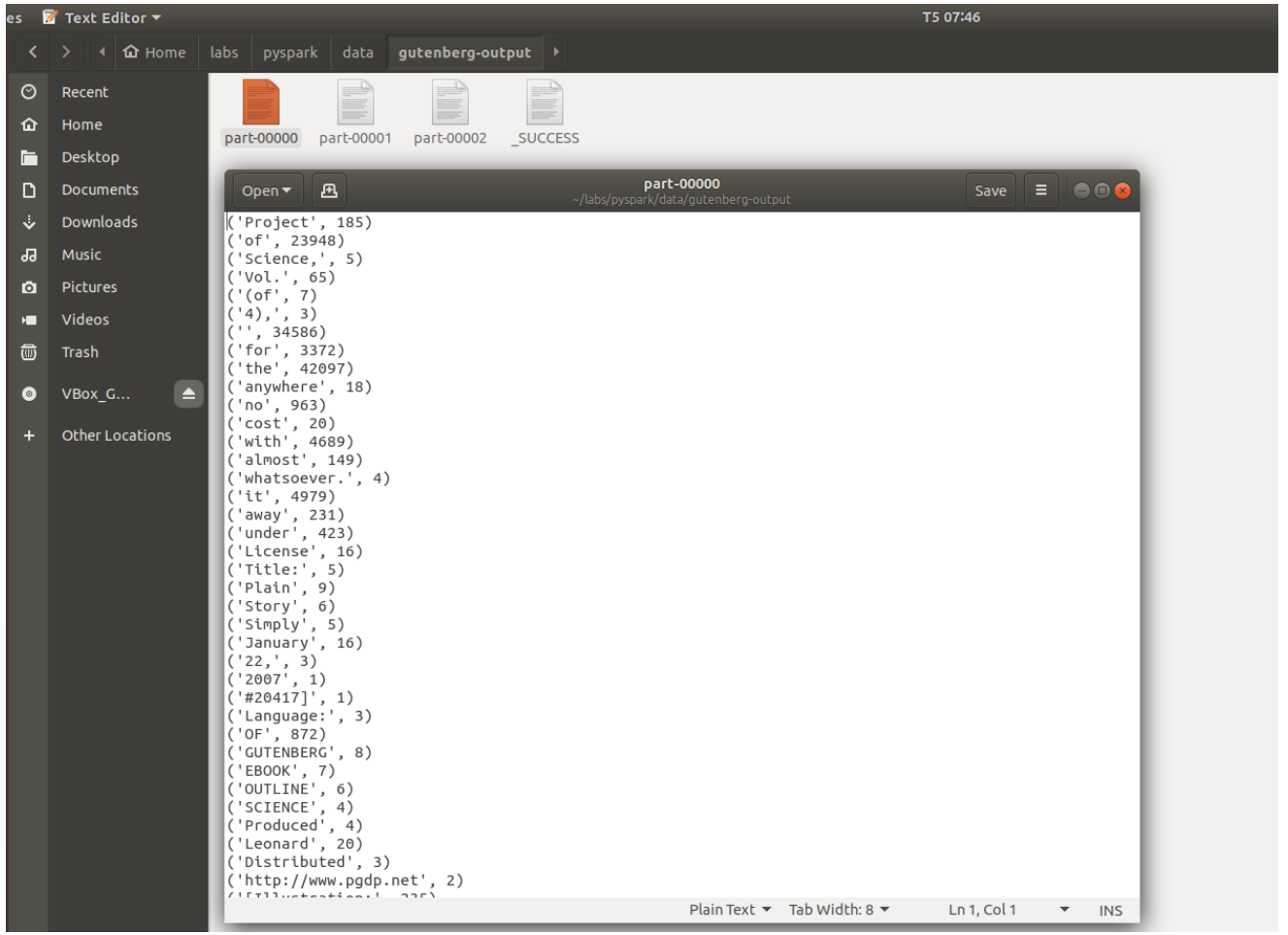
PySpark – Word Count example

```
import sys

from pyspark import SparkContext, SparkConf

if __name__ == "__main__":
    # create Spark context with necessary configuration
    sc = SparkContext("local", "PySpark Word Count Example")

    # read data from text file and split each line into words
    words = sc.textFile("/home/hung/labs/pyspark/data/gutenberg/") \
        .flatMap(lambda line: line.split(" "))

    # count the occurrence of each word
    wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b)

    # save the counts to output
    wordCounts.saveAsTextFile("/home/hung/labs/pyspark/data/gutenberg-output/")
```

PySpark – Word Count example

- Run

```
spark-submit wordcount.py
```



Lập trình với RDDs

- Resilient distributed datasets (RDDs) là thành phần cốt lõi của Spark xử lý dữ liệu
- Các tác vụ trên Spark đều liên quan đến RDDs:
  - Tạo mới RDDs
  - Chuyển đổi RDDs
  - Thực thi xử lý tính toán trên RDDs
- Spark tự động thực hiện phân tán dữ liệu trong RDDs và tiến hành xử lý song song trên cluster.

Căn bản về RDD

- Một RDD là một tập các đối tượng phân tán. Mỗi RDD được chia thành nhiều phân vùng, mỗi phân vùng được xử lý trên các máy trạm khác nhau.
- RDDs có thể chứa các kiểu đối tượng Python, Java, Scala hoặc của lớp tự định nghĩa
- Có thể tạo RDDs bằng hai cách:
  - Nạp dữ liệu từ bên ngoài:

    Ví dụ: Tạo RDD các dòng văn bản với PySpark (nạp từ bên ngoài)

    >>> lines = sc.textFile("/home/hung/Downloads/gutenberg")
  - Tạo từ trình điều khiển

    Ví dụ: Tạo RDD các xâu ký tự

    >>> lines = sc.parallelize(["pandas", "i like pandas"])

Căn bản về RDD

- Sau khi RDDs được tạo ra, có thể thực hiện hai loại xử lý trên nó: *transformations* and *actions*.
- *Transformations (biến đổi):* Tạo RDD mới từ RDD trước đó.

  Ví dụ: Lọc các dòng thông báo lỗi trong file log sử dụng biến đổi filter() :

  >>> errorLines = lines.filter(**lambda** line: "error" **in** line)
- *Actions (hành động):* Xử lý lấy kết quả từ RDD và trả về cho trình điều khiển hoặc lưu lên bộ nhớ ngoài

  Ví dụ: Lấy ra dòng báo lỗi đầu tiên sử dụng action first():

  >>> pythonLines.first()

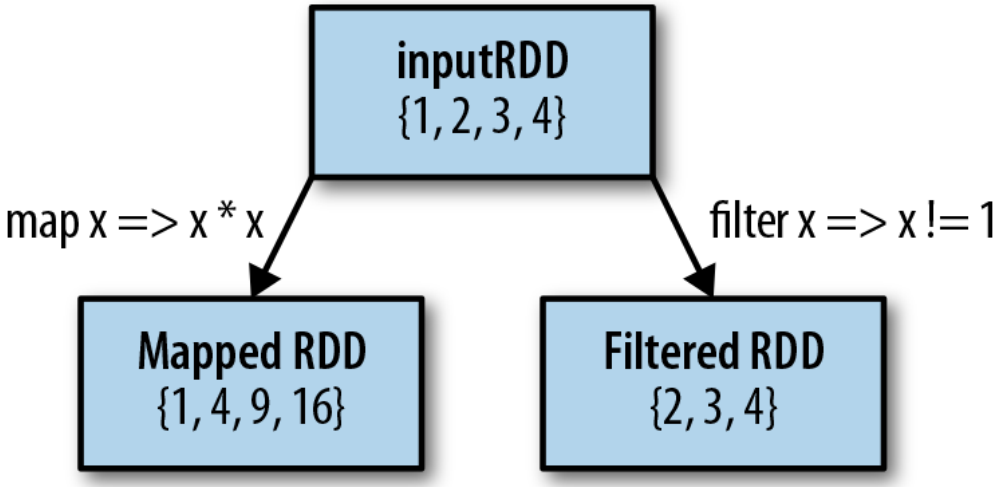
Các transformations thông dụng

- map() transformation

Nhận một hàm và gọi hàm lên từng thành phần của RDD, kết quả trả về của hàm trở thành giá trị mới của mỗi thành phần của RDD kết quả.

- filter() transformation

Nhận một hàm, trả về RDD chứa các thành phần thỏa mãn hàm filter()



Các transformations thông dụng

- Có thể dùng biến đổi map() thực hiện nhiều loại xử lý:
  - Tính toán trên dữ liệu (vd, tính bình phương của dãy số)
  - Lấy trang web từ URL
  - …
- Lưu ý:
  - Kiểu trả về của biến đổi map() không bắt buộc cùng kiểu với dữ liệu vào

  Ví dụ: PySpark bình phương dãy số trong RDD

```
nums = sc.parallelize([1, 2, 3, 4])
squared = nums.map(lambda x: x * x).collect()
for num in squared:
print("%i " % (num))
```

Các transformations thông dụng

- flatMap()

Với mỗi giá trị đầu vào, trả về nhiều gía trị

Ví dụ: tách tập văn bản ra thành các từ

```
lines = sc.parallelize(["hello world", "hi"])
words = lines.flatMap(lambda line: line.split(" "))
words.first() # returns "hello"
```

Các actions thông dụng

- reduce()

Nhận vào một hàm thao tác với 2 phần tử trên RDD, trả về phần tử mới cùng kiểu.

Ví dụ: Tính tổng các phần tử với PySpark:

```
sum = rdd.reduce(lambda x, y: x + y)
```

- collect()

Trả về nội dung của RDD

- take(n)

Trả về *n* phần tử từ RDD

- count()

Trả về tổng số phần tử trong RDD

- countByValue()

Trả về số lượng mỗi phần tử trong RDD

Xử lý các cặp Key/Value

- Trên RDDs, các cặp Key/value thường dùng cho các thao tác gộp dữ liệu (aggregation)
- Dữ liệu vào được chuyển thành các cặp key/value thông qua thao tác ETL (extract, transform, load)
- Các cặp Key/value được xử lý theo mục đích bài toán (vd: đếm số reviews của các sản phẩm, nhóm dữ liệu cùng <key> ,…)

Tạo cặp Key/value RDDs

- Có thể dùng hàm map() để tạo các cặp key/value trên RDD.
- Ví dụ: Tạo một RDD từ dữ liệu văn bản và lấy từ đầu tiên mỗi dòng làm khóa:

```
pairs = lines.map(lambda x: (x.split(" ")[0], x))
```

Biến đổi cặp Key/value

- Các cặp Key/value RDDs có kiểu tuple, vì thế cần gọi hàm thao tác trên tuple thay vì trên từng phần tử riêng lẻ.
- reduceByKey(): chạy các xử lý reduce song song, kết hợp các giá trị có cùng khóa, trả về RDD chứa các khóa và giá trị rút gọn kèm theo.