

25 YEARS ANNIVERSARY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Chương 6

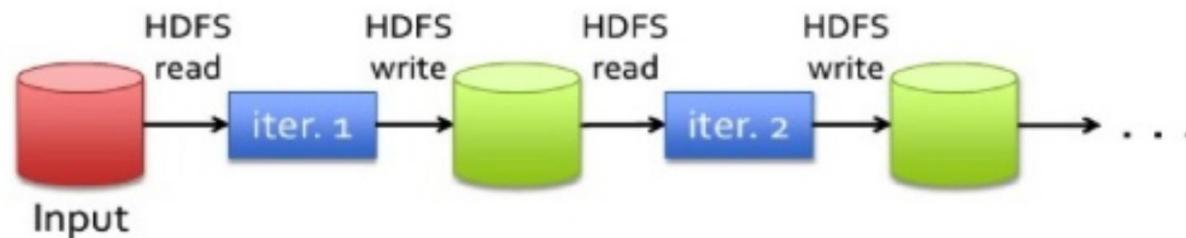
Xử lý hàng loạt - phần 2

Tia lửa Apache

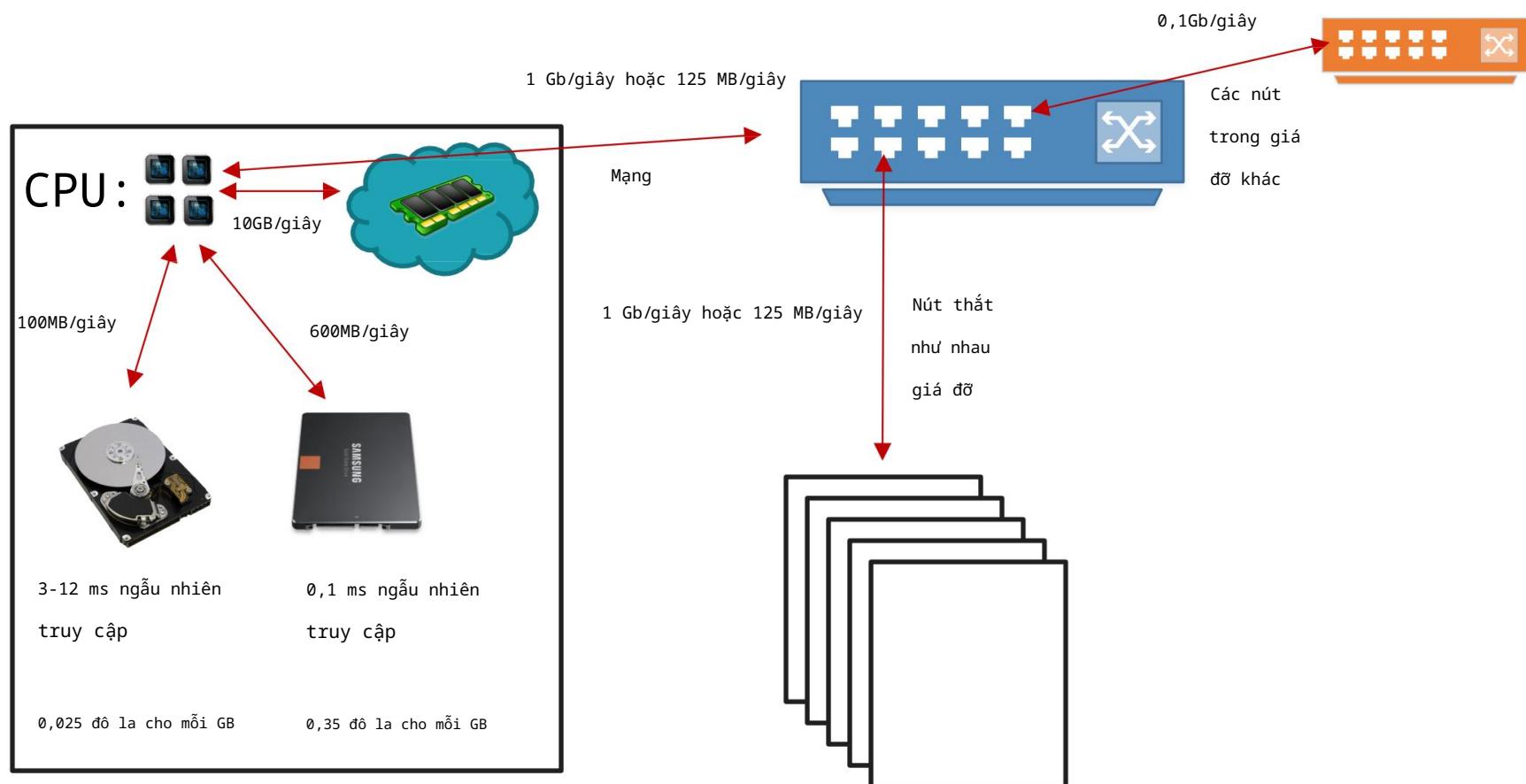
Một công cụ phân tích thống nhất để xử lý dữ liệu
quy mô lớn

Map Reduce: Các công việc lặp lại

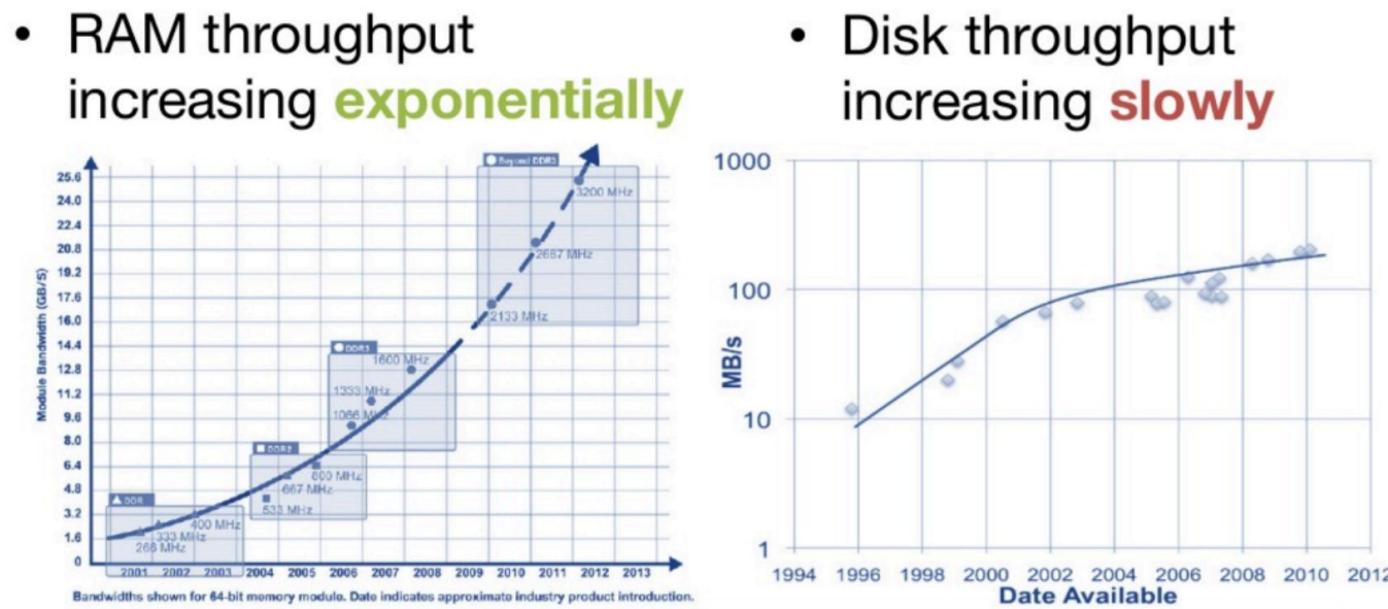
- Các công việc lặp đi lặp lại liên quan đến rất nhiều I/O đĩa cho mỗi lần lặp lại



- è Đĩa I/O rất chậm!



RAM là ổ đĩa mới

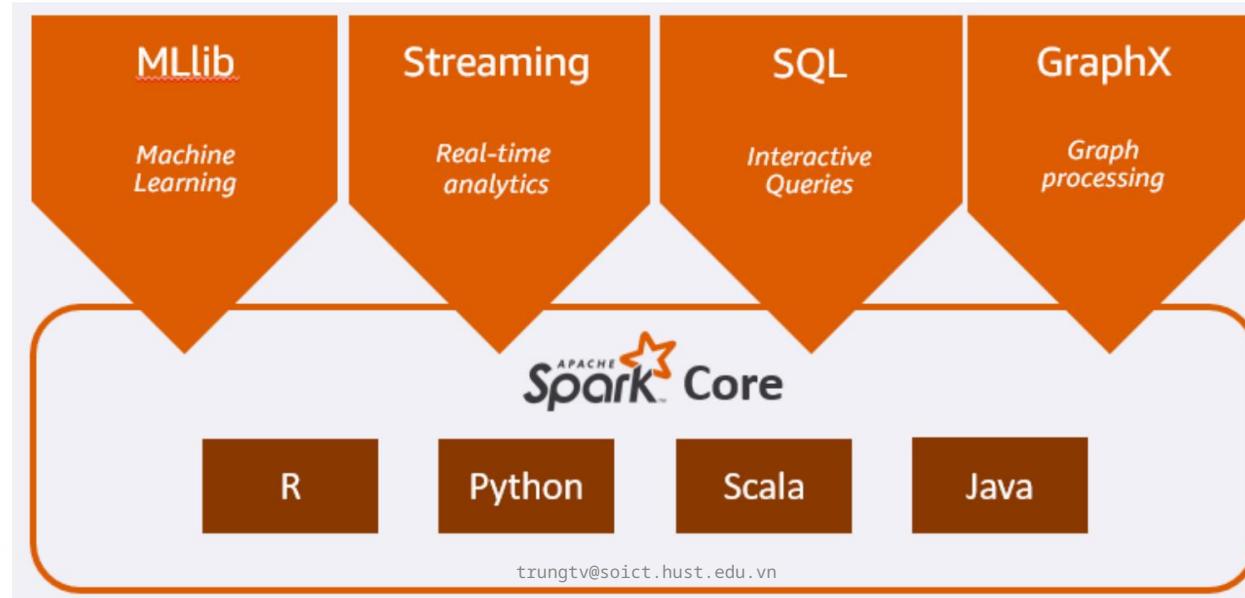


Memory-locality key to interactive response times

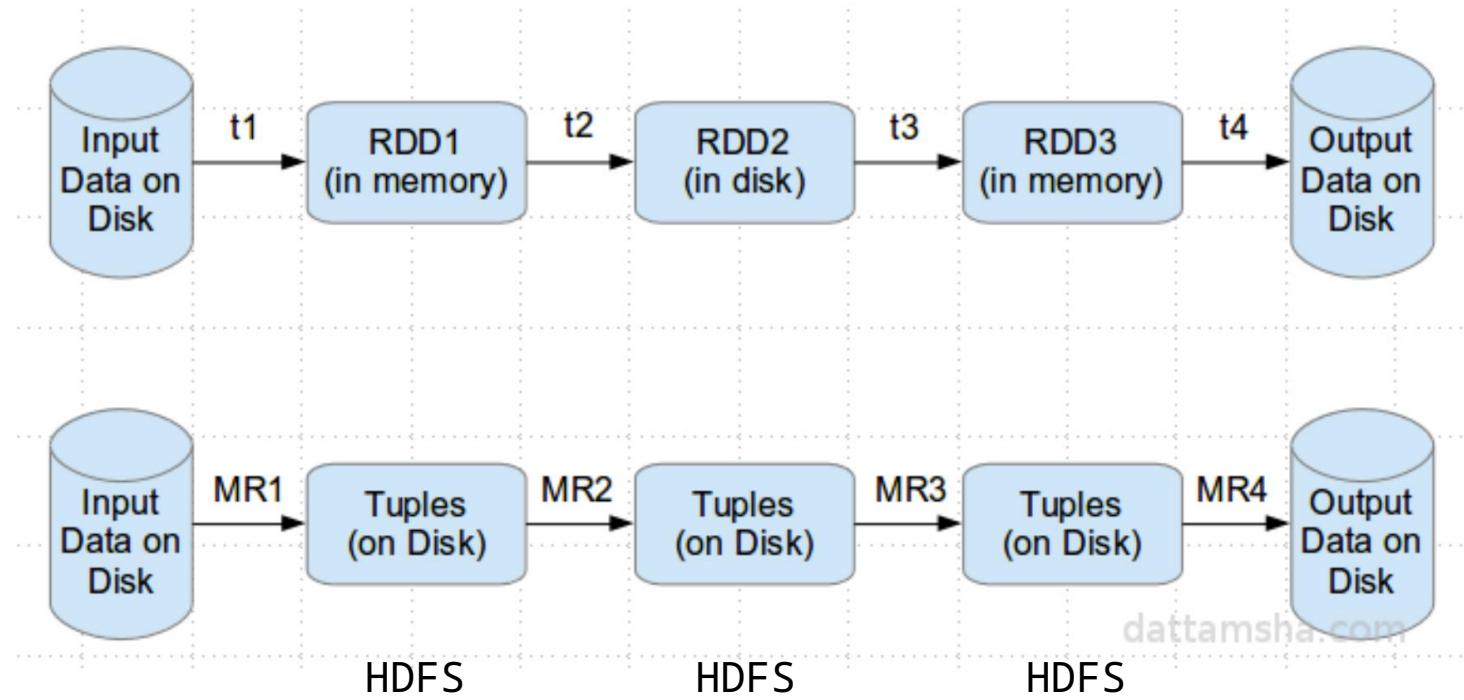
Một công cụ phân tích thống nhất để xử lý dữ liệu quy mô lớn

- Hỗ trợ tốt hơn cho
 - Thuật toán lặp lại •
- Khai thác dữ liệu tương tác
- Khả năng chịu lỗi, vị trí dữ liệu, khả năng mở rộng •
- Ẩn

các phức tạp: giúp người dùng tránh việc mã hóa để cấu trúc cơ chế phân tán.



Bộ nhớ thay vì đĩa



Spark và Map Giảm sự khác biệt

	Apache Hadoop MR	Tia lửa Apache
Kho	Chỉ đĩa	Trong bộ nhớ hoặc trên đĩa
Hoạt động	Bản đồ và Giảm	Nhiều chuyển đổi và hành động, bao gồm cả Map và Reduce
Mô hình thực hiện Lô		Hàng loạt, lặp lại, phát trực tuyến
Ngôn ngữ	Java	Scala, Java, Python và R

Apache Spark so với Apache Hadoop

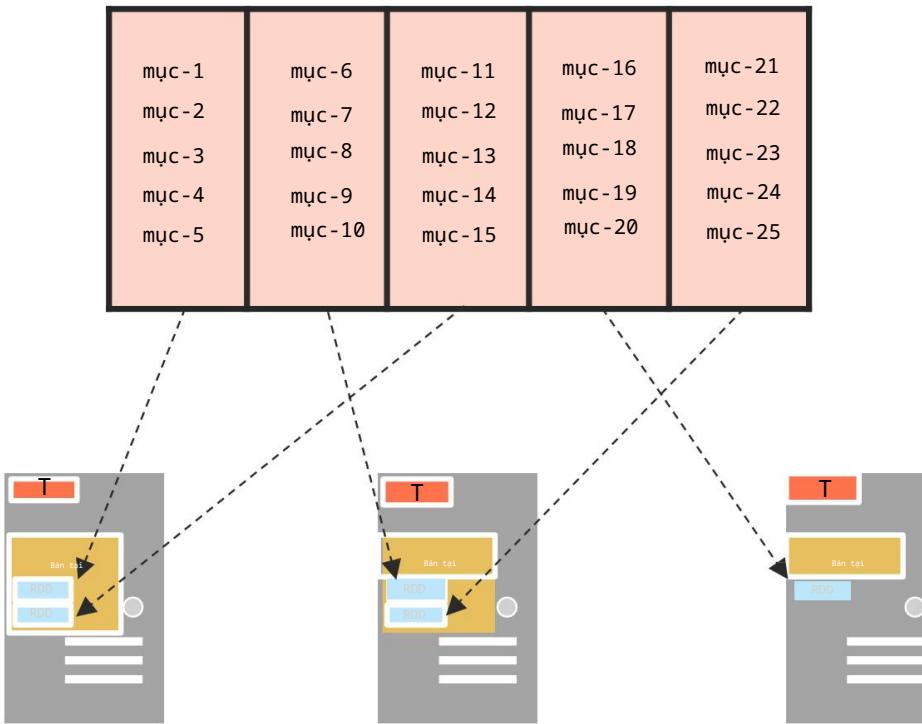
	Hadoop World Record	Spark 100 TB *	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Sort Benchmark	Yes	Yes	No
Daytona Rules			
Environment	dedicated data center	EC2 (i2.8xlarge)	EC2 (i2.8xlarge)

Bộ dữ liệu phân tán có khả năng phục hồi (RDD)

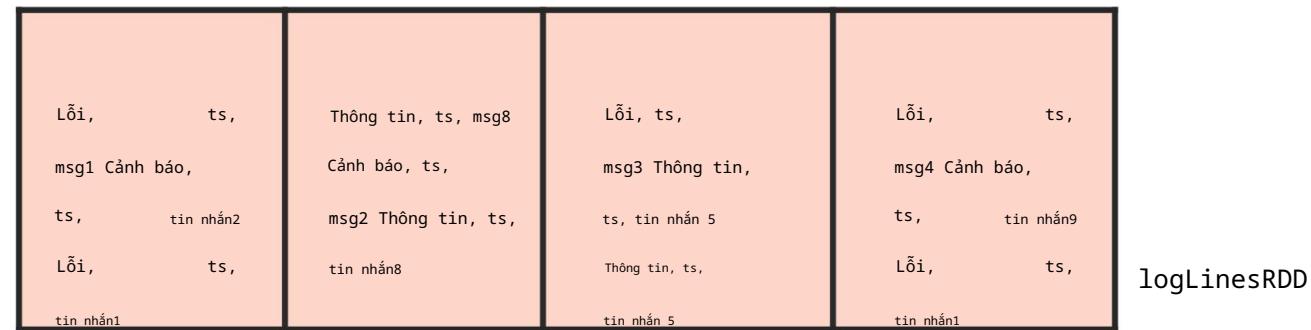
- RDD là cấu trúc dữ liệu song song, có khả năng chịu lỗi , cho phép người dùng lưu trữ rõ ràng các kết quả trung gian trong bộ nhớ, kiểm soát phân vùng của chúng để tối ưu hóa vị trí dữ liệu và thao tác chúng bằng một tập hợp toán tử phong phú.
- chuyển đổi hạt thô so với hạt mịn cập nhật
 - ví dụ, ánh xạ, lọc và nối) áp dụng cùng một thao tác cho nhiều mục dữ liệu cùng một lúc.

nhiều phân vùng = nhiều tính song song hơn

RDD



RDD với 4 phân vùng



Có thể tạo RDD cơ sở theo 2 cách:

- Song song hóa một bộ sưu tập
- Đọc dữ liệu từ nguồn bên ngoài (S3, C*, HDFS, v.v.)

Song song hóa



```
// Song song hóa trong Scala  
val wordsRDD = sc.parallelize(Danh sách("cá", "mèo", "chó"))
```

- Lấy một bộ sưu tập trong bộ nhớ hiện có và chuyển nó tới phương thức parallelize của SparkContext

```
# Song song hóa trong Python  
wordsRDD = sc.parallelize(["cá", "mèo", "chó"])
```

- Không được sử dụng chung bên ngoài việc tạo mẫu và thử nghiệm vì nó yêu cầu toàn bộ tập dữ liệu trong bộ nhớ trên một máy



```
// Song song hóa trong Java  
JavaRDD<String> wordsRDD = sc.parallelize(Arrays.asList("cá", "mèo", "chó"));
```



Đọc từ tệp văn bản



```
// Đọc tệp txt cục bộ trong Scala  
val linesRDD = sc.textFile("/đường dẫn/đến/README.md")
```

Có những phương pháp khác để đọc dữ liệu từ HDFS, C*, S3, HBase, v.v.



```
# Đọc tệp txt cục bộ trong Python  
linesRDD = sc.textFile("/đường dẫn/đến/README.md")
```

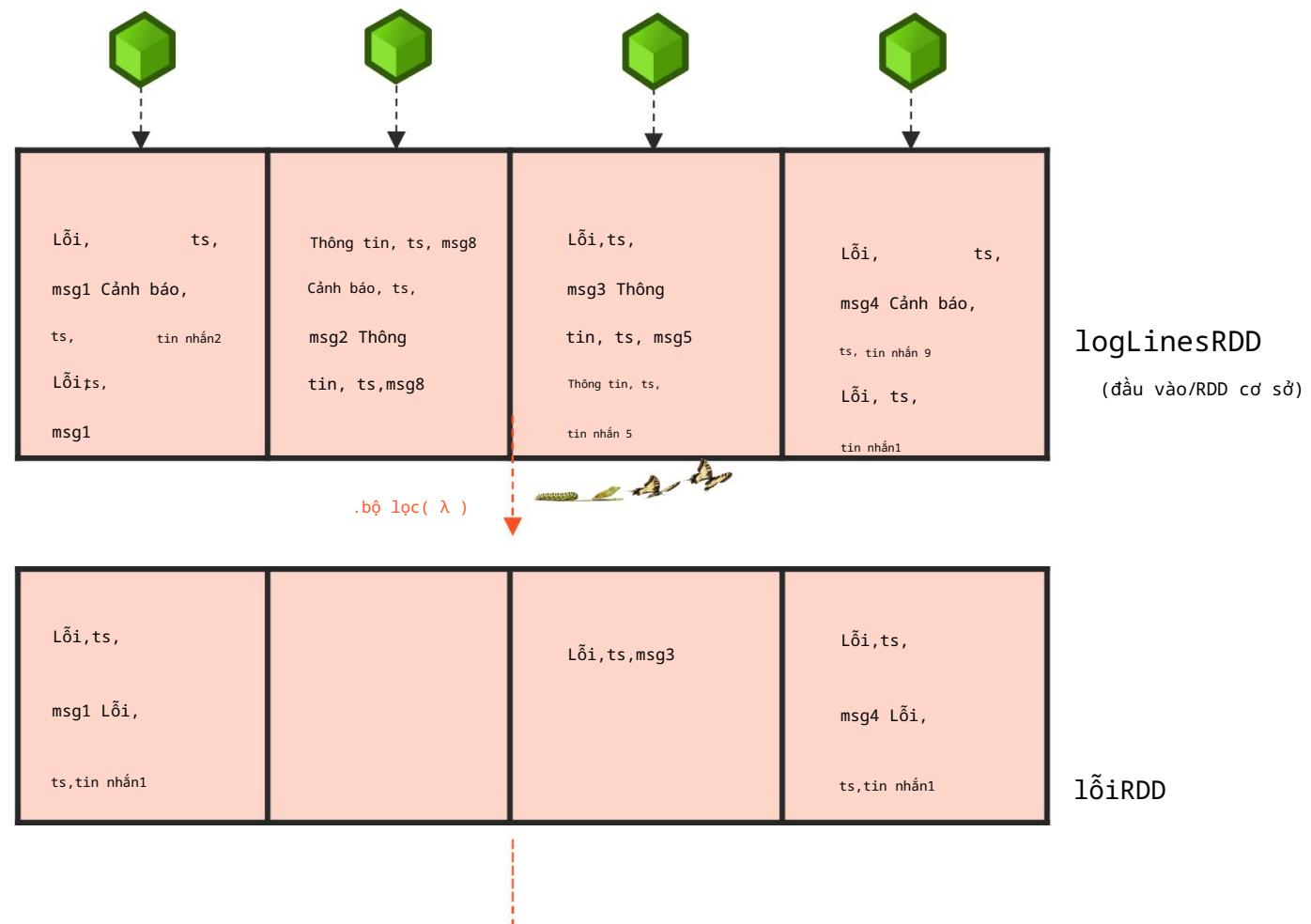


```
// Đọc tệp txt cục bộ trong Java  
JavaRDD<String> dòng = sc.textFile("/đường dẫn/đến/README.md");
```

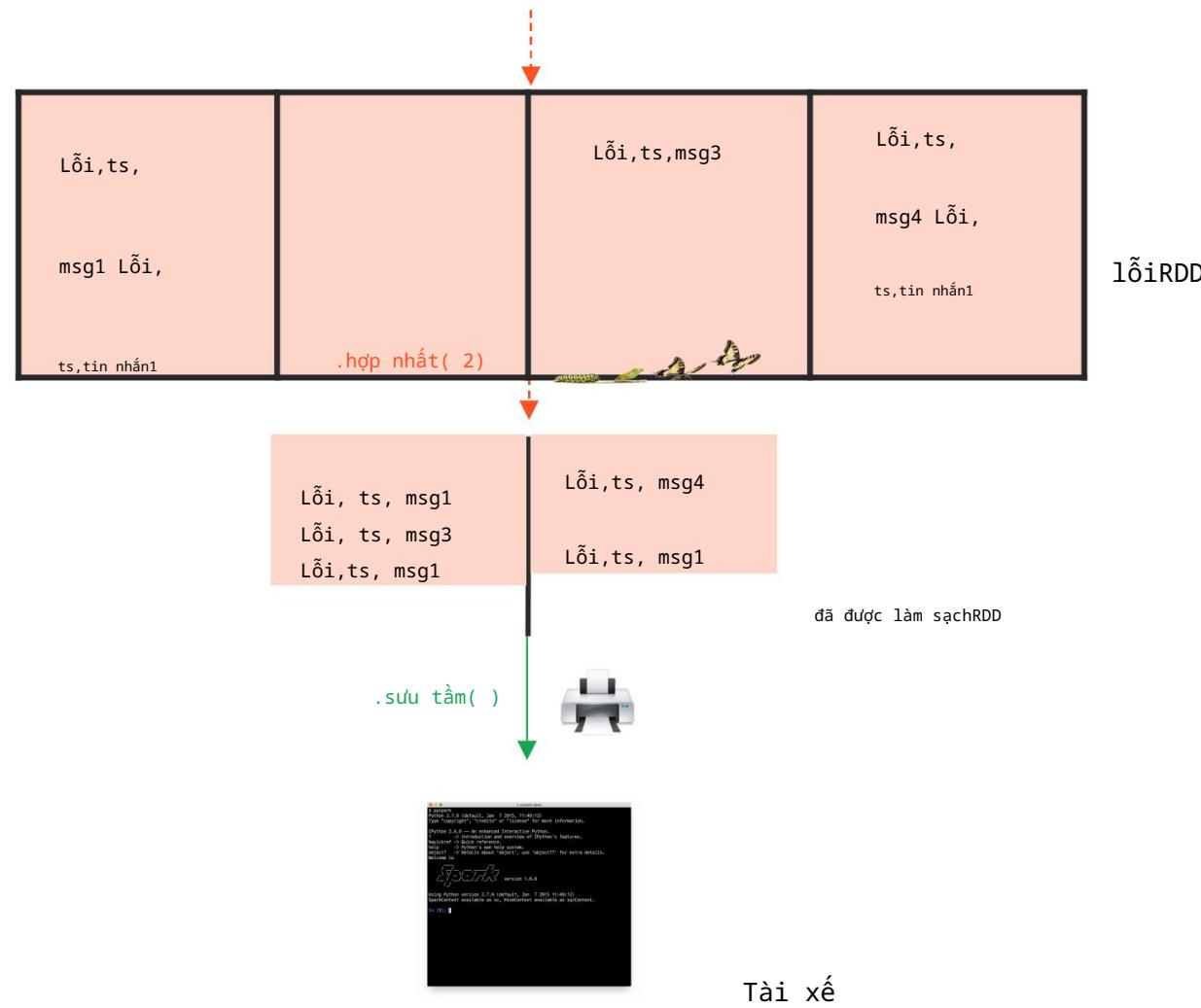
Các hoạt động trên dữ liệu phân tán

- Hai loại hoạt động: chuyển đổi và hành động • Chuyển đổi là lười biếng (không được tính toán ngay lập tức) • Chuyển đổi được thực hiện khi một hành động được chạy
- Lưu trữ (bộ nhớ đệm) dữ liệu phân tán trong bộ nhớ hoặc đĩa

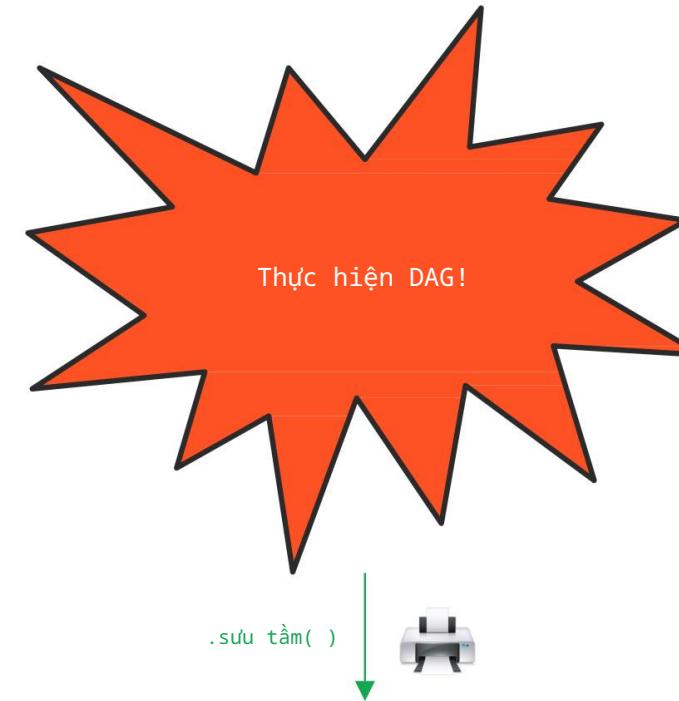
Chuyển đổi: Bộ lọc



Hành động: Thu thập

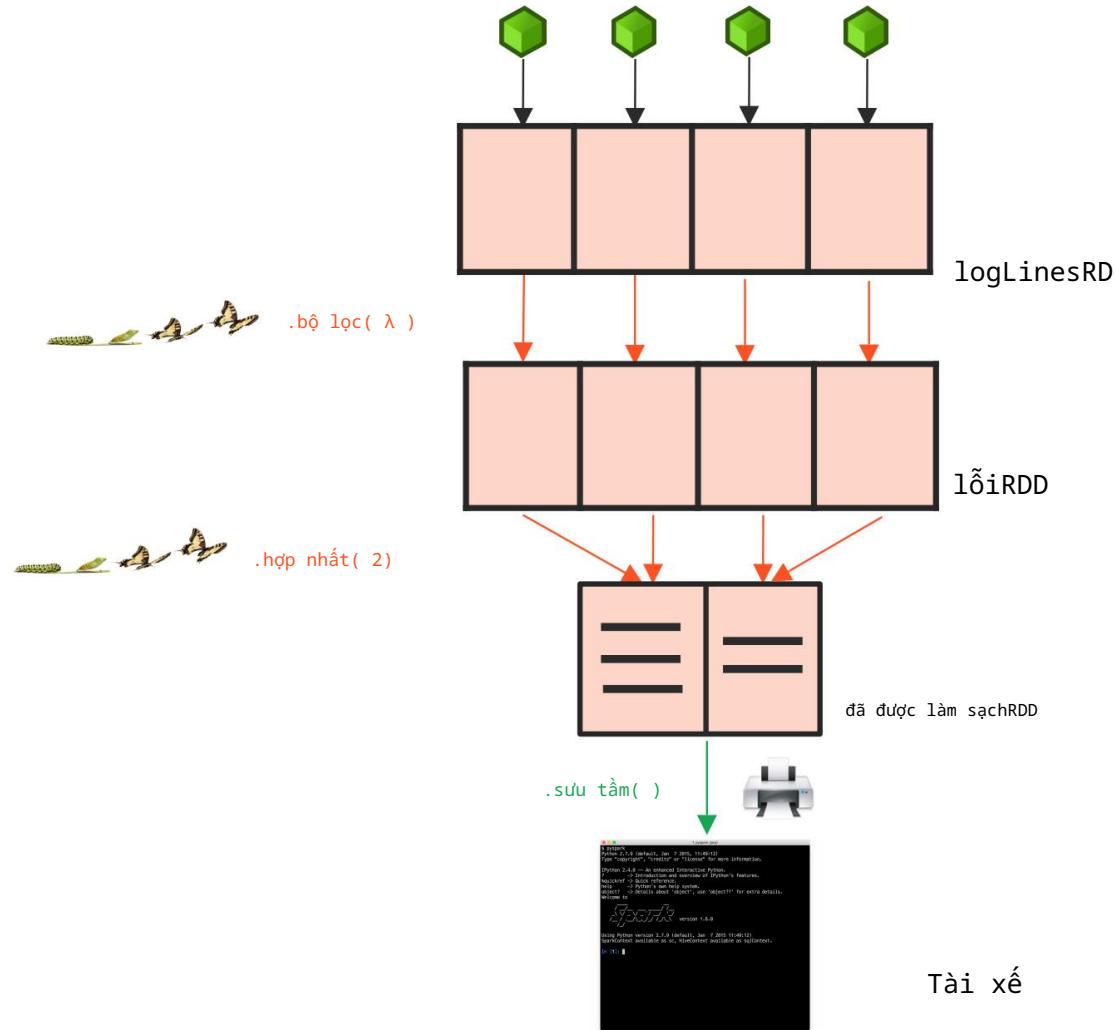


Thực hiện DAG

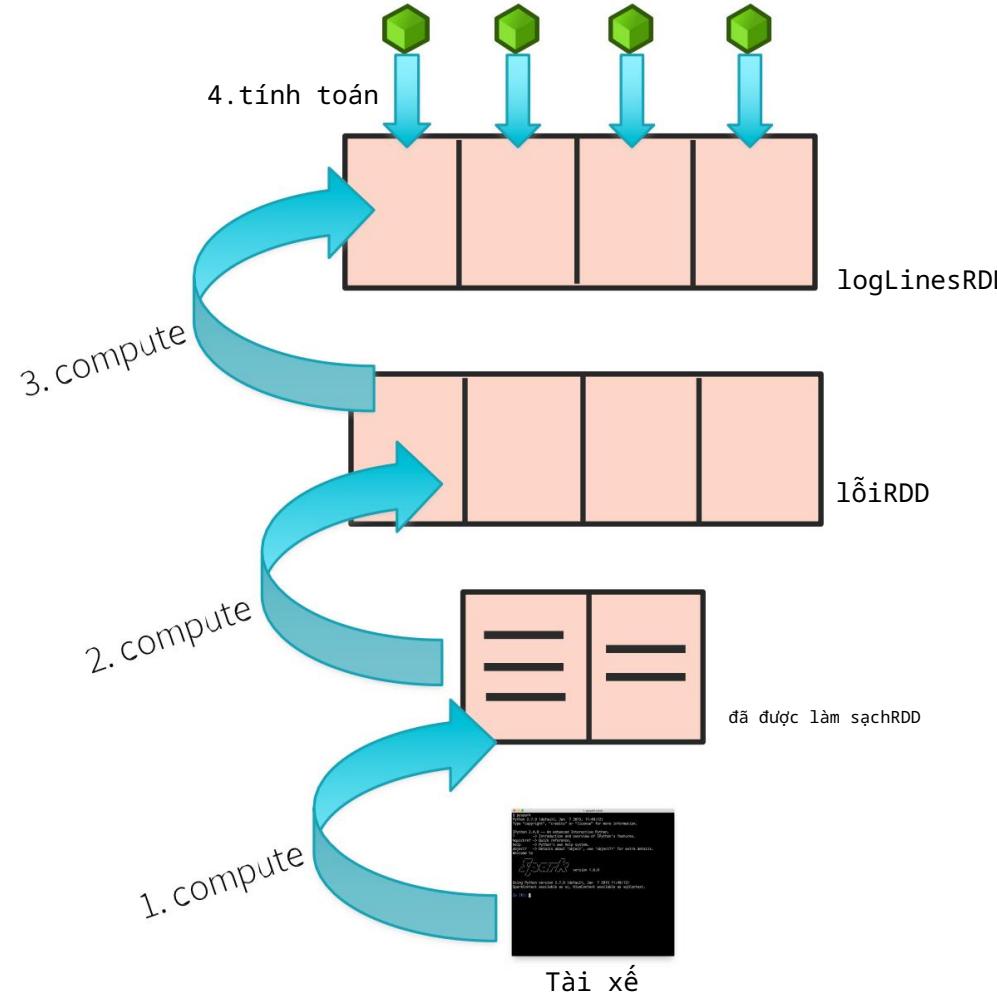


Tài xé

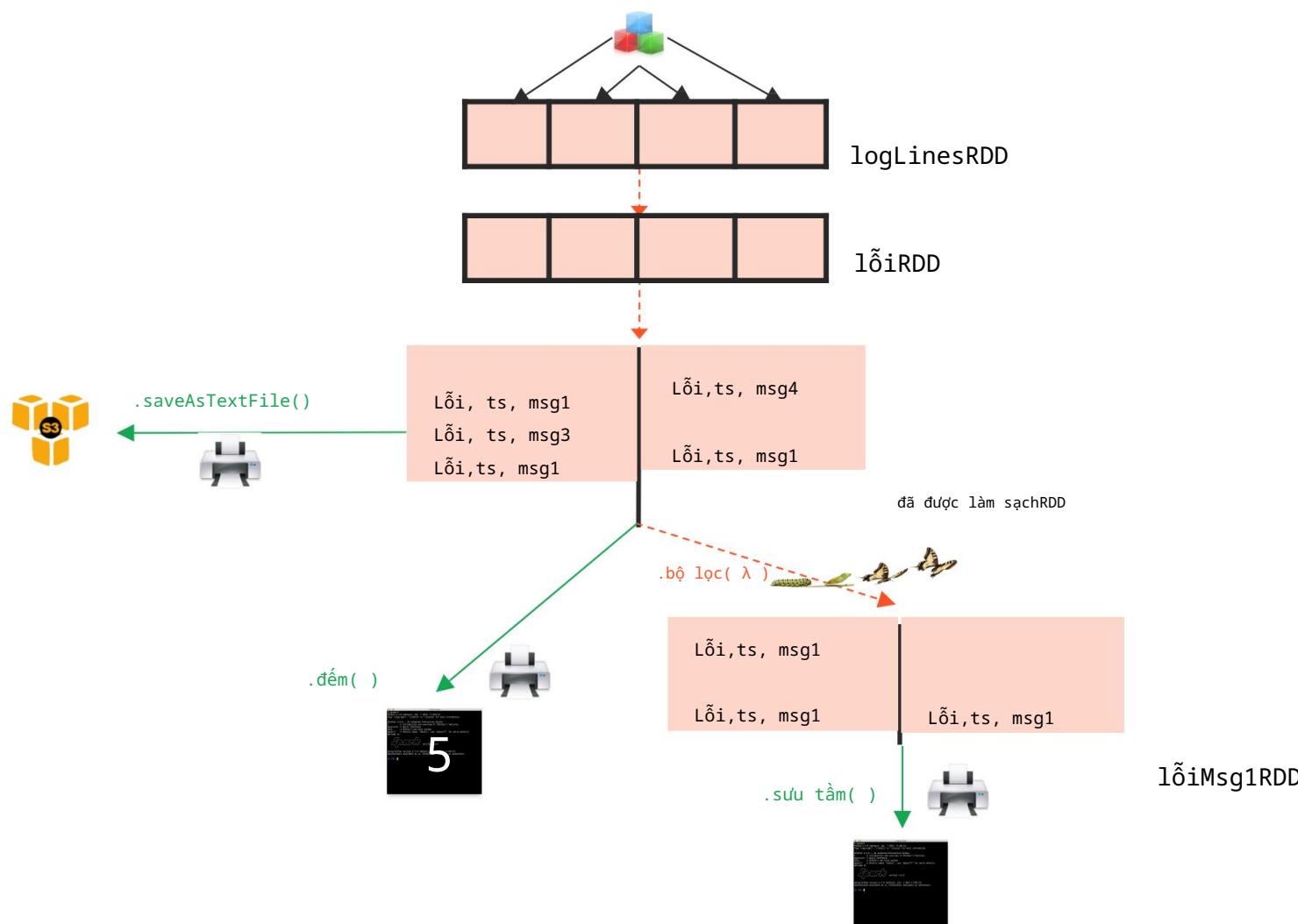
Hợp lý



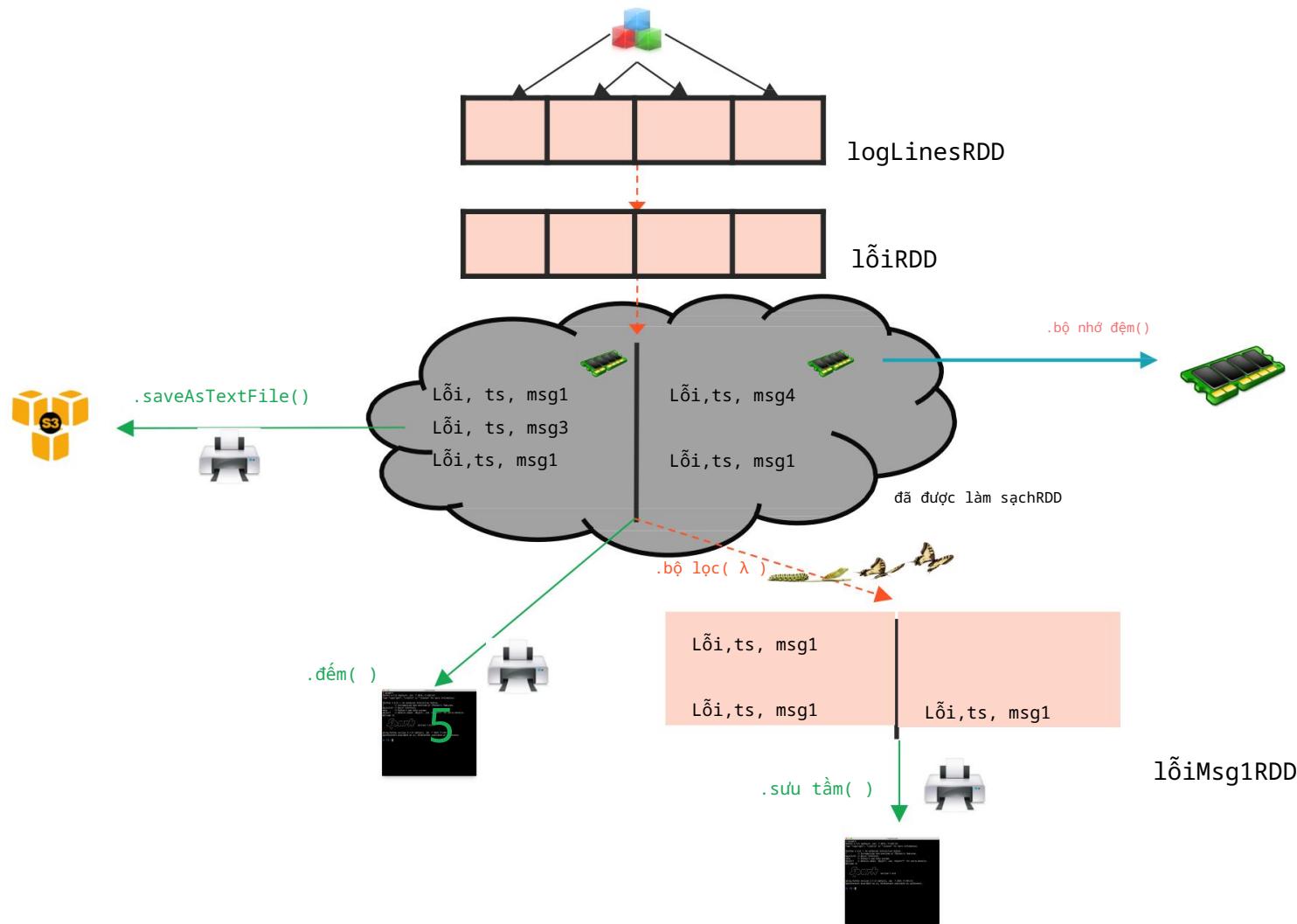
Thuộc vật chất



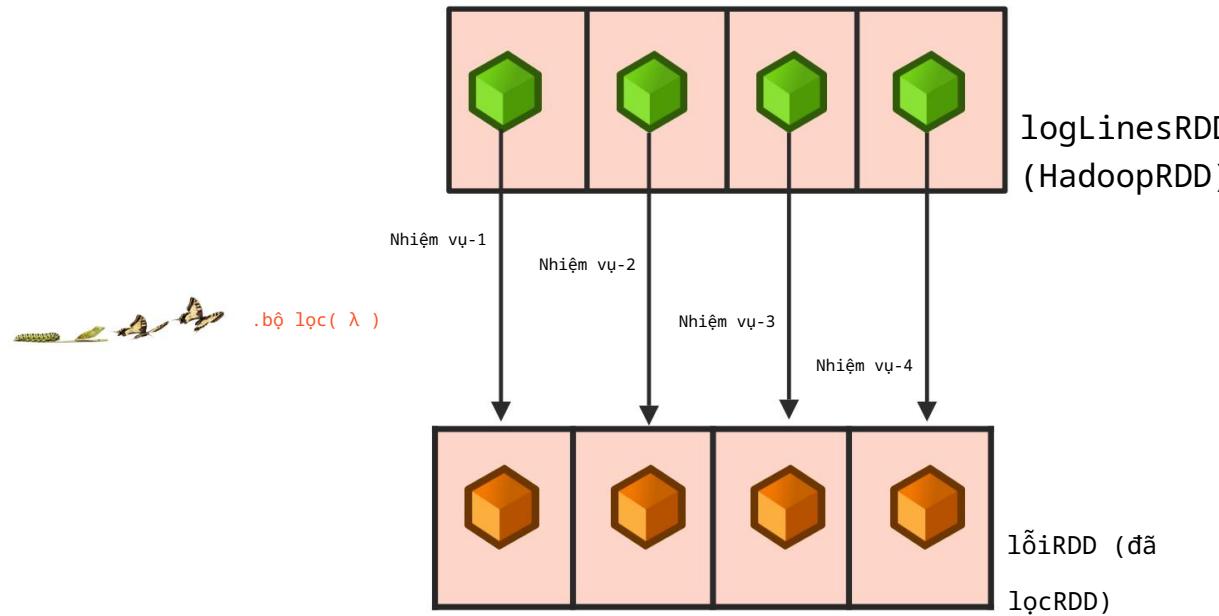
DAG



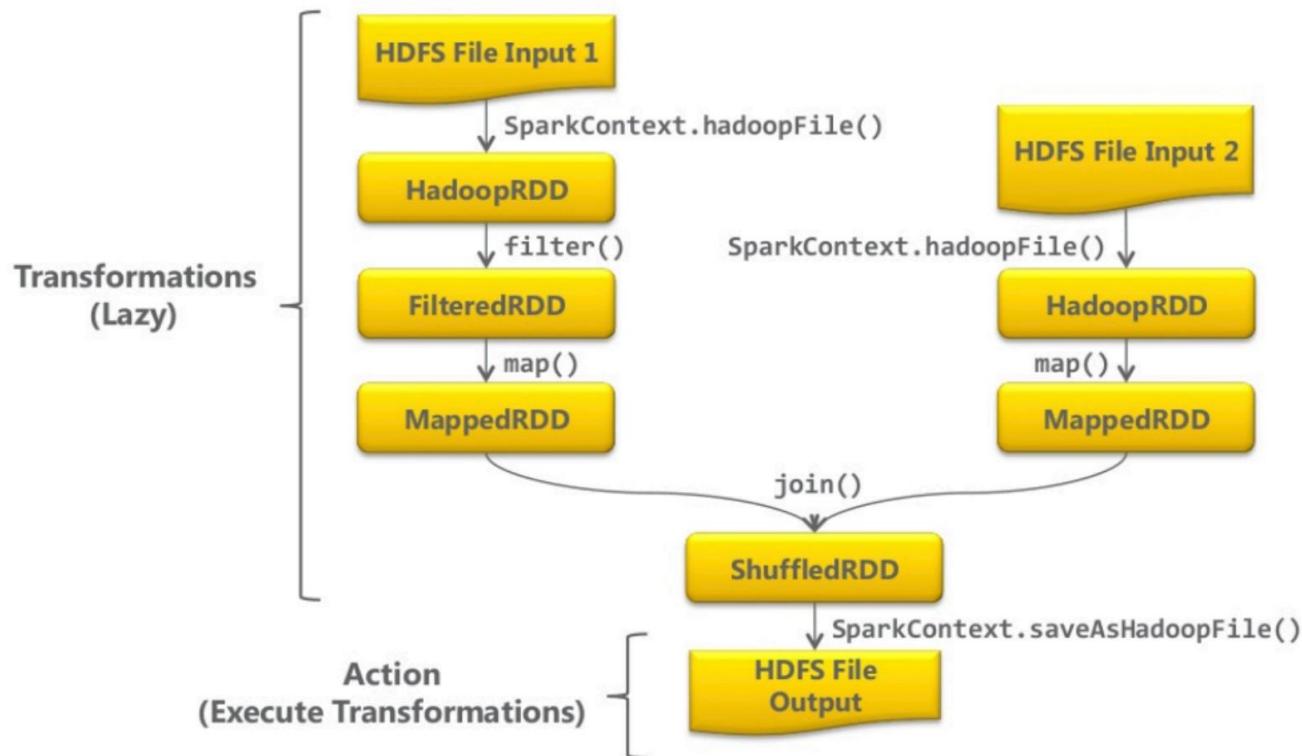
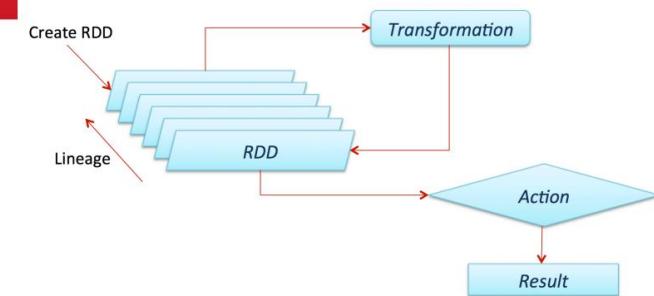
Bộ nhớ đệm



Phân vùng >>> Nhiệm vụ >>> Phân vùng



Dòng dõi RDD



Bộ dữ liệu phân tán có khả năng phục hồi (RDD)

- RDD ban đầu trên đĩa (HDFS, v.v.) • RDD trung gian trên RAM
- Phục hồi lỗi dựa trên dòng dõi • Hoạt động RDD được phân phối

Khung dữ liệu

- Một sự trừu tượng chính trong Spark 2.0
 - Không thay đổi sau khi xây dựng • Theo dõi thông tin dòng dõi để tính toán lại dữ liệu đã mất một cách hiệu quả • Cho phép thực hiện các hoạt động thu thập các phần tử song song
- Để xây dựng DataFrame
 - Bằng cách song song hóa các bộ sưu tập Python hiện có (danh sách)
 - Bằng cách chuyển đổi một Spark hoặc pandas DataFrame hiện có • Từ các tệp trong HDFS hoặc hệ thống lưu trữ khác

Sử dụng DataFrame

```
>>> data = [('Alice', 1), ('Bob', 2), ('Bob', 2)]  
>>> df1 = sqlContext.createDataFrame(dữ liệu, ['tên', 'tuổi'])  
  
[Hàng(tên=u'Bob', tuổi=2),  
 Hàng(tên=u'Bob', tuổi=2),  
 Hàng(tên=u'Bob', tuổi=2)]
```

Biến đổi

- Tạo DataFrame mới từ một DataFrame hiện có
- Sử dụng đánh giá lười biếng
 - Không có gì thực thi • Spark lưu công thức cho nguồn chuyển đổi

Sự biến đổi	Sự miêu tả
chọn(*cột)	Chọn các cột từ DataFrame này
thả(cột)	Trả về một DataFrame mới xóa cột cụ thể
bộ lọc(hàm)	Trả về một DataFrame mới được hình thành bằng cách chọn những hàng của nguồn mà func trả về giá trị true
nơi(func)	Bí danh cho bộ lọc ở đâu?
distinct()	Trả về một DataFrame mới chứa các hàng riêng biệt của nguồn DataFrame
sắp xếp(*cột, **kw)	Trả về một DataFrame mới được sắp xếp theo các cột được chỉ định và theo thứ tự sắp xếp được chỉ định bởi kw



Sử dụng phép biến đổi

```
>>> data = [('Alice', 1), ('Bob', 2), ('Bob', 2)]
```

```
>>> df1 = sqlContext.createDataFrame(dữ liệu, ['tên', 'tuổi'])
```

```
>>> df2 = df1.distinct()
```

```
[Hàng(tên=u'Alice', tuổi=1), Hàng=(tên=u'Bob', tuổi=2)]
```

```
>>> df3 = df2.sort("tuổi", tăng dần=False)
```

```
[Hàng=(tên=u'Bob', tuổi=2), Hàng(tên=u'Alice', tuổi=1)]
```

Hành động

- Khiến Spark thực hiện công thức để chuyển đổi nguồn
- Cơ chế để có được kết quả từ Spark

Hoạt động	Sự miêu tả
show(n, cắt bớt)	In n hàng đầu tiên của DataFrame này
take(n)	Trả về n hàng đầu tiên dưới dạng danh sách Hàng
collect()	Trả về tất cả các bản ghi dưới dạng danh sách Row (*)
count()	Trả về số hàng trong DataFrame này
mô tả(*cột)	Hàm Phân tích dữ liệu thăm dò tính toán số liệu thống kê (đếm, trung bình, độ lệch chuẩn, min, max) cho các cột số

Sử dụng Hành động

```
>>> data = [('Alice', 1), ('Bob', 2)]  
>>> df = sqlContext.createDataFrame(dữ liệu, ['tên', 'tuổi'])  
>>> df.thu thập()  
[Hàng(tên=u'Alice', tuổi=1), Hàng=(tên=u'Bob', tuổi=2)]  
>>> df.đếm()  
2  
>>> df.hiển thị()  
+-----+-----+  
|tên|  tuổi |  
+-----+-----+  
|Alice|      1|  
|Bob |      2|  
+-----+-----+
```

Bộ nhớ đệm

```
>>> linesDF = sqlContext.read.text('..')  
>>> linesDF.cache()  
>>> commentsDF = linesDF.filter(isComment)  
>>> in linesDF.count(), commentsDF.count()  
>>> commentsDF.cache()
```

Quy trình lập trình Spark

- Tạo DataFrames từ dữ liệu bên ngoài hoặc
createDataFrame từ một bộ sưu tập trong chương trình điều khiển
- Biến đổi chúng một cách lười biếng thành DataFrames mới
- cache() một số DataFrame để tái sử dụng
- Thực hiện các hành động để thực hiện tính toán song song và tạo
ra kết quả

DataFrames so với RDD

- Đối với người dùng mới quen thuộc với các khung dữ liệu trong các ngôn ngữ lập trình, API này sẽ khiến họ cảm thấy như ở nhà
- Đối với người dùng Spark hiện tại, API sẽ giúp Spark dễ lập trình hơn so với sử dụng RDD
- Đối với cả hai nhóm người dùng, DataFrames sẽ cải thiện hiệu suất thông qua tối ưu hóa thông minh và tạo mã

Viết ít mã hơn: Đầu vào & Đầu ra

Giao diện thống nhất để đọc/ghi dữ liệu ở nhiều định dạng khác nhau.

```
giá trị df = sqlContext.  
          đọc.định dạng("json").  
          tùy chọn("tỷ lệ lấy mẫu", "0,1").  
          tải("/Người dùng/spark/dữ liệu/stuff.json")  
  
df. viết.  
      format("parquet").  
      mode("append").  
      partitionBy("year").  
      saveAsTable("faster-stuff")
```



Viết ít mã hơn: Đầu vào & Đầu ra

Giao diện thống nhất để đọc/ghi dữ liệu ở nhiều định dạng khác nhau.

```
val df =  
    sqlContext.read.format("json").  
    tùy chọn("tỷ lệ lấy mẫu", "0,1").  
    tải("/Người dùng/spark/dữ liệu/stuff.json")  
  
df. viết.  
    format("parquet").  
    mode("append").  
    partitionBy("year").  
    saveAsTable("faster-stuff")
```

đọc và ghi các hàm
tạo ra các trình xây
dựng mới để thực
hiện I/O

Viết ít mã hơn: Đầu vào & Đầu ra

Giao diện thống nhất để đọc/ghi dữ liệu ở nhiều định dạng khác nhau.

```
val df = sqlContext.đã
đọc.

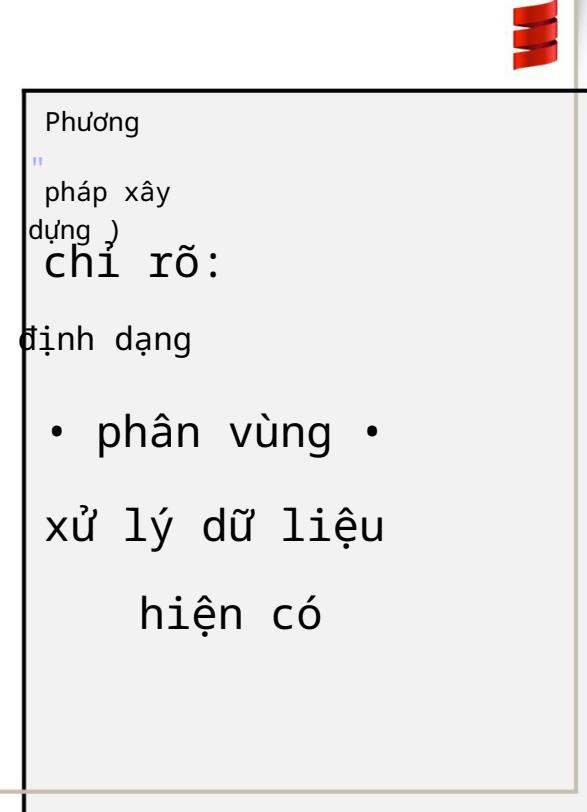
định dạng ("json").

}tùy chọn("Tỷ lệ lấy mẫu", "0,1").
tải("/Users/spark/data/stuff.json" • df. viết.

chế độ ("thêm").

}format("sàn gỗ").

partitionBy("năm").
saveAsTable("nhanh hơn
-thứ")
```



Viết ít mã hơn: Đầu vào & Đầu ra

Giao diện thống nhất để đọc/ghi dữ liệu ở nhiều định dạng khác nhau.

```
val df =  
  
    sqlContext.read.format ("json").  
    tùy chọn("tỷ lệ lấy mẫu", "0,1").  
    Users/spark/data/stuff.json")  
  
df. viết.  
    format("sàn gỗ").  
    mode("thêm vào").  
    partitionBy("năm").  
    saveAsTable("faster-stuff")
```

tải(.), lưu(.), tải("/
hoặc saveAsTable(.) kết
thúc I/O
đặc điểm kỹ thuật

Nguồn dữ liệu được hỗ trợ bởi DataFrames

được xây dựng sẵn



bên ngoài



và nhiều hơn nữa.

Viết ít mã hơn: Cấp cao Hoạt động

- Giải quyết các vấn đề phổ biến một cách ngắn gọn với DataFrame chức năng:
 - chọn cột và lọc
 - kết nối các nguồn dữ liệu khác nhau
 - tổng hợp (đếm, tổng, trung bình, v.v.)
 - vẽ biểu đồ kết quả (ví dụ, với Pandas)

Viết ít mã hơn: Tính toán trung bình



```

private IntWritable mót = new IntWritable(1); private
IntWritable đầu ra =new IntWritable(); protected void
map(LongWritable key,
     Giá trị văn bản,
     Bối cảnh bối cảnh) {
    String[] fields = value.split("\t"); đầu
    ra.set(Integer.parseInt(fields[1])); context.write(one,
    đầu ra);
}

.....
IntWritable mót = new IntWritable(1)
DoubleWritable trung bình = new DoubleWritable();

được bảo vệ void reduce(IntWritable key,
     Giá trị <IntWritable> có thể lặp lại,
     Bối cảnh bối cảnh) {
    int tổng = 0;
    int đếm = 0; đôi
    với (giá trị IntWritable: giá trị) { tổng += giá
    trị.get(); đếm++;

    } average.set(tổng / (double) count);
    context.write(khóa, trung bình);
}

```



```

rdd = sc.textFile(...).map(_.split(" ")).reduceByKey { trường hüp ((số1,
(x0), (x1.toFloat, 1))}. reduceByKey { trường hüp ((số1,
số đếm1), (số2, số đếm2)) =>
    (số1 + số2, số đếm1 + số đếm2)

}. map { case (key, (num, count)) => (key, num / count) }. collect()

```



```

rdd = sc.textFile(...).map(lambda s: s.split()).reduceByKey(lambda
x: (x[0], (float(x[1]), 1))).map(lambda t1, t2:
    (t1[0] + t2[0], t1[1] + t2[1])).map(lambda t: (t[0], t[1][0] / t[1][1])).collect()

```



Viết ít mã hơn: Tính toán một Trung bình

Sử dụng RDDs

```
rdd = sc.textFile(...).map(_.split(" ")).rdd.map
{ x => (x(0), (x(1).toFloat, 1)) }.reduceByKey
{ trường hợp ((số1, số đếm1), (số2, số đếm2)) =>
  (số1 + số2, số1 + số2) }.map { trường
  hợp (khóa, (số, số)) => (khóa, số / số) }.collect()
```



Tài liệu API

đầy đủ •

[Scala](#) •

[Java](#) •

[Python](#) • [R](#)

Sử dụng DataFrames

```
nhập org.apache.spark.sql.functions._

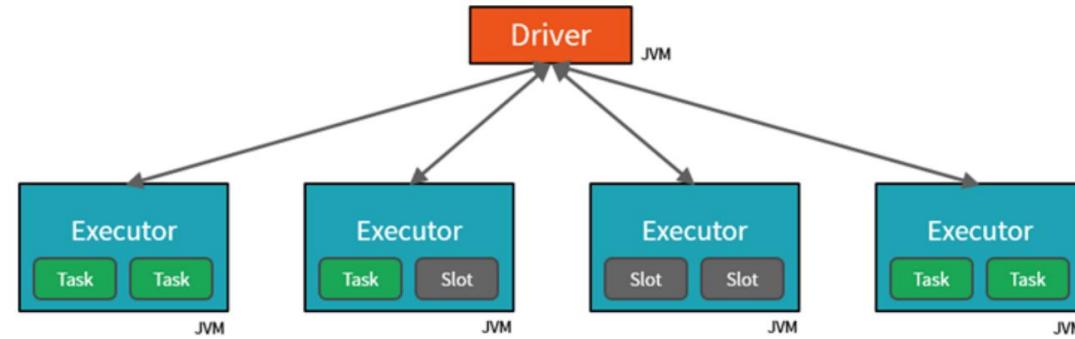
val df = rdd.map(a => (a(0), a(1))).toDF("key", "value").df.groupBy("key")

.agg(avg("giá
trị")).collect()
```



Ngành kiến trúc

- Kiến trúc kiểu chủ-thợ
 - Một trình điều khiển hoặc nút chính
 - Các nút công nhân



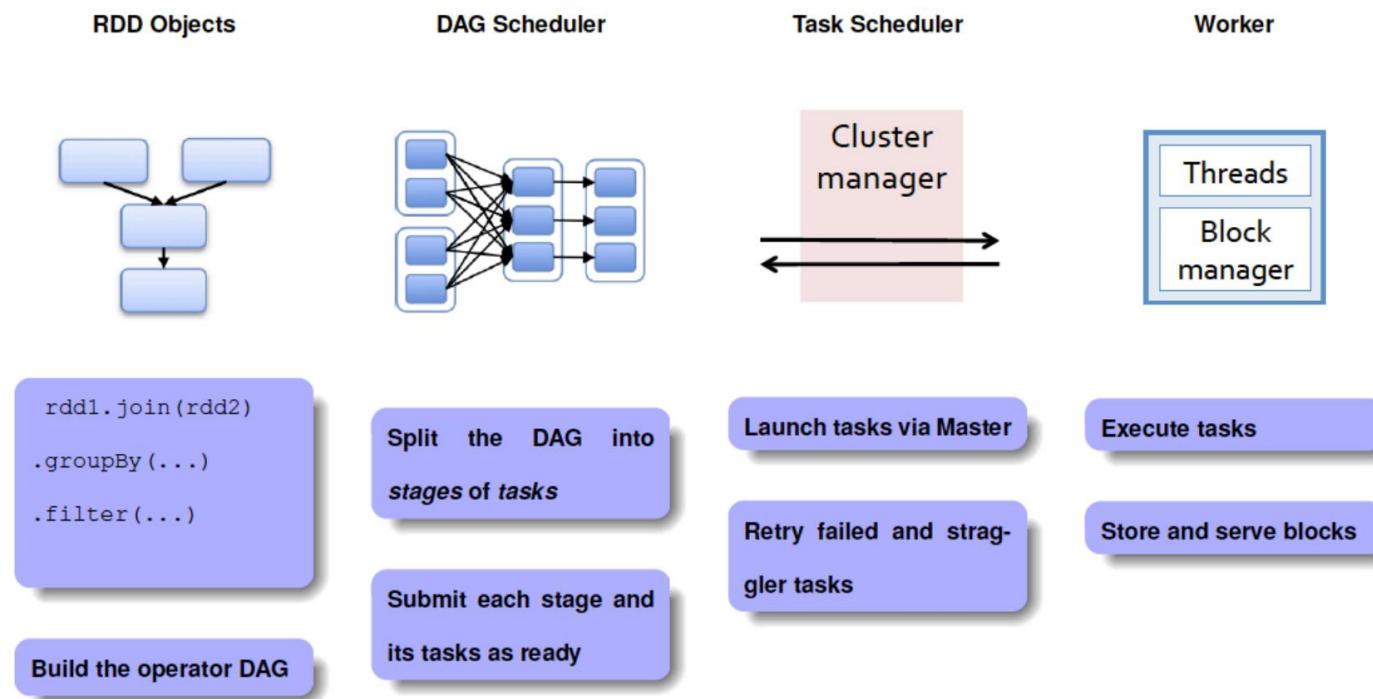
- Người chủ giao việc cho công nhân và hướng dẫn họ lấy dữ liệu từ bộ nhớ hoặc từ ổ cứng (hoặc từ nguồn khác như S3 hoặc HDFS)

Kiến trúc(2)

- Một chương trình Spark đầu tiên tạo ra một đối tượng SparkContext
 - SparkContext cho Spark biết cách thức và nơi truy cập vào cụm
 - Tham số chính cho SparkContext xác định loại và kích thước cụm nào sẽ sử dụng

Tham số chính	Sự miêu tả
địa phương	Chạy Spark cục bộ với một luồng công nhân (không song song)
địa phương[K]	Chạy Spark cục bộ với K luồng công nhân (lý tưởng nhất là thiết lập số lõi)
spark://HOST:CỘNG	Kết nối với cụm Spark độc lập
mesos://HOST:CỘNG	Kết nối với cụm Mesos
sợi len	Kết nối với cụm YARN

Thời gian làm việc tại Spark



Thử nghiệm



SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Tài liệu tham khảo

- Zaharia, Matei, et al. "Các tập dữ liệu phân tán có khả năng phục hồi: Một lỗi trừu tượng hóa dung sai cho máy tính cụm trong bộ nhớ." Được trình bày như một phần của Hội nghị chuyên đề {USENIX} lần thứ 9 về Thiết kế và Triển khai Hệ thống Mạng ({NSDI} 12). 2012.
- Armbrust, Michael, et al. "Spark sql: Xử lý dữ liệu quan hệ trong spark." Biên bản báo cáo hội nghị quốc tế ACM SIGMOD năm 2015 về quản lý dữ liệu. 2015.
- Zaharia, Matei, et al. "Các luồng rời rạc: Chịu lỗi tính toán phát trực tuyến ở quy mô lớn." Biên bản hội nghị chuyên đề ACM lần thứ hai mươi tư về nguyên tắc hệ điều hành. 2013.
- Chambers, Bill và Matei Zaharia. Spark: Hướng dẫn xác đáng: Xử lý dữ liệu lớn trở nên đơn giản. "O'Reilly Media, Inc.", 2018.



25
YEARS ANNIVERSARY
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Cảm ơn sự
chú ý
của bạn!!!

