1. Spark properties

Spark properties kiểm soát hầu hết các cài đặt ứng dụng và được cấu hình riêng cho từng ứng dụng. Các thuộc tính này có thể được đặt trực tiếp trên SparkConf được chuyển tới SparkContext của bạn. SparkConf cho phép bạn định cấu hình một số thuộc tính phổ biến (ví dụ: URL chính và tên ứng dụng), cũng như các cặp khóa-giá trị tùy ý thông qua phương thức set(). Ví dụ: chúng ta có thể khởi tạo một ứng dụng với hai luồng như sau:

Lưu ý rằng chúng tôi chạy với local [2], nghĩa là hai luồng - thể hiện sự song song "tối thiểu", có thể giúp phát hiện lỗi chỉ tồn tại khi chúng tôi chạy trong bối cảnh phân tán.

```
val conf = new SparkConf()
             .setMaster("local[2]")
             .setAppName("CountingSheep")
val sc = new SparkContext(conf)
```

Lưu ý rằng chúng ta có thể có nhiều hơn 1 luồng ở chế độ cục bộ và trong những trường hợp như Spark Streaming, chúng tôi thực sự có thể yêu cầu nhiều hơn 1 luồng để ngăn chặn bất kỳ loại vấn đề chết đói nào. Các thuộc tính chỉ định một số khoảng thời gian nên được cấu hình với một đơn vị thời gian. Định dạng sau được chấp nhận:

```
val conf = new SparkConf()
25ms (milliseconds)
5s (seconds)
10m or 10min (minutes)
3h (hours)
5d (days)
1y (years)
```

Thuộc tính chỉ định kích thước byte phải được cấu hình với đơn vị kích thước. Định dạng sau được chấp nhận:

```
1b (bytes)
1k or 1kb (kibibytes = 1024 bytes)
1m or 1mb (mebibytes = 1024 kibibytes)
1g or 1gb (gibibytes = 1024 mebibytes)
1t or 1tb (tebibytes = 1024 gibibytes)
1p or 1pb (pebibytes = 1024 tebibytes)
```

Trong khi các số không có đơn vị thường được hiểu là byte, một số ít được hiểu là KiB hoặc MiB. Xem tài liệu về các thuộc tính cấu hình riêng lẻ. Việc chỉ định đơn vị là mong muốn nếu có thể.

Dynamically Loading Spark Properties

Trong một số trường hợp, bạn có thể muốn tránh mã hóa cứng các cấu hình nhất định trong SparkConf. Ví dụ: nếu bạn muốn chạy cùng một ứng dụng với các bản chính khác nhau hoặc số lượng bộ nhớ khác nhau. Spark cho phép bạn chỉ cần tạo một conf trống:

```
val sc = new SparkContext(new SparkConf())
```

Sau đó, bạn có thể cung cấp các giá trị cấu hình trong thời gian chạy:

```
./bin/spark-submit --name "My app" --master local[4] --conf spark.eventLog.enabled=false
  --conf "spark.executor.extraJavaOptions=-XX:+PrintGCDetails -XX:+PrintGCTimeStamps" myApp.jar
```

Spark shell và công cụ spark-submit hỗ trợ hai cách để tải cấu hình động. Đầu tiên là các tùy chọn dòng lệnh, chẳng hạn như --master, như hình trên. spark-submit có thể chấp nhận bất kỳ thuộc tính Spark nào sử dụng cờ --conf / -c, nhưng sử dụng cờ đặc biệt cho các thuộc tính đóng một vai trò trong việc khởi chạy ứng dụng Spark. Chạy ./bin/spark-submit --help sẽ hiển thị toàn bộ danh sách các tùy chọn này. bin / spark-submit cũng sẽ đọc các tùy chọn cấu hình từ conf / spark-defaults.conf, trong đó mỗi dòng bao gồm một khóa và một giá trị được phân tách bằng khoảng trắng. Ví dụ:

```
spark.master            spark://5.6.7.8:7077
spark.executor.memory   4g
spark.eventLog.enabled  true
spark.serializer        org.apache.spark.serializer.KryoSerializer
```

Mọi giá trị được chỉ định dưới dạng cờ hoặc trong tệp thuộc tính sẽ được chuyển đến ứng dụng và được hợp nhất với những giá trị được chỉ định thông qua SparkConf. Các thuộc tính được đặt trực tiếp trên SparkConf được ưu tiên cao nhất, sau đó các cờ được chuyển đến spark-submit hoặc spark-shell, sau đó là các tùy chọn trong tệp spark-defaults.conf. Một vài khóa cấu hình đã được đổi tên kể từ các phiên bản Spark trước đó; trong những trường hợp như vậy, các tên khóa cũ hơn vẫn được chấp nhận, nhưng được ưu tiên thấp hơn bất kỳ trường hợp nào của khóa mới hơn.

Các thuộc tính của Spark chủ yếu có thể được chia thành hai loại: một là liên quan đến triển khai, như “spark.driver.memory”, “spark.executor.instances”, loại thuộc tính này có thể không bị ảnh hưởng khi thiết lập lập trình thông qua SparkConf trong thời gian chạy, hoặc hành vi tùy thuộc vào trình quản lý cụm và chế độ triển khai bạn chọn, vì vậy bạn nên đặt thông qua tệp cấu hình hoặc tùy chọn dòng lệnh spark-submit; một loại khác chủ yếu liên quan đến kiểm soát thời gian chạy Spark, như “spark.task.maxFailures”, loại thuộc tính này có thể được đặt theo một trong hai cách.

Viewing Spark Properties

Giao diện người dùng web ứng dụng tại http: // <driver>: 4040 liệt kê các thuộc tính Spark trong tab "Environment". Đây là một nơi hữu ích để kiểm tra để đảm bảo rằng các thuộc tính của bạn đã được đặt chính xác. Lưu ý rằng chỉ các giá trị được chỉ định rõ ràng thông qua spark-defaults.conf, SparkConf hoặc dòng lệnh mới xuất hiện. Đối với tất cả các thuộc tính cấu hình khác, bạn có thể giả sử giá trị mặc định được sử dụng.

Available Properties

Hầu hết các thuộc tính kiểm soát cài đặt nội bộ đều có giá trị mặc định hợp lý. Một số tùy chọn phổ biến nhất để đặt là:

Property Name	Default	Meaning	Since Version
spark.app.name	(none)	Tên ứng dụng của bạn. Điều này sẽ xuất hiện trong giao diện người dùng và trong log data	0.9.0
spark.driver.cores	1	Số lõi để sử dụng cho quy trình trình điều khiển, chỉ ở chế độ cụm.	1.3.0
spark.driver.maxResultSize	1g	Giới hạn tổng kích thước của các kết quả được tuần tự hóa của tất cả các phân vùng cho mỗi hành động Spark (ví dụ: thu thập) tính bằng byte. Tối thiểu phải là 1M hoặc 0 cho không giới hạn. Công việc sẽ bị hủy bỏ nếu tổng kích thước vượt quá giới hạn này. Có giới hạn cao có thể gây ra lỗi hết bộ nhớ trong trình điều khiển (phụ thuộc vào spark.driver.memory và chi phí bộ nhớ của các đối tượng trong JVM). Đặt giới hạn thích hợp có thể bảo vệ trình điều khiển khỏi lỗi hết bộ nhớ.	1.2.0
spark.driver.memory	1g	Dung lượng bộ nhớ sẽ sử dụng cho quá trình trình điều khiển, tức là nơi SparkContext được khởi tạo, có định dạng giống như chuỗi bộ nhớ JVM với hậu tố đơn vị kích thước ("k", "m", "g" hoặc "t") (ví dụ: 512m, 2g). Lưu ý: Ở chế độ máy khách, cấu hình này không được đặt thông qua	1.1.1

		SparkConf trực tiếp trong ứng dụng của bạn, vì trình điều khiển JVM đã bắt đầu tại thời điểm đó. Thay vào đó, hãy đặt điều này thông qua tùy chọn dòng lệnh --driver-memory hoặc trong tệp thuộc tính mặc định của bạn.	
spark.driver.memoryOverhead	driverMemory * 0.10, with minimum of 384	Số lượng bộ nhớ không phải heap sẽ được cấp phát cho mỗi quá trình trình điều khiển ở chế độ cụm, trong MiB trừ khi được chỉ định khác. Đây là bộ nhớ chiếm những thứ như tổng chi phí VM, chuỗi được thực hiện, các chi phí chung khác, v.v. Điều này có xu hướng phát triển theo kích thước vùng chứa (thường là 6-10%). Tùy chọn này hiện được hỗ trợ trên YARN, Mesos và Kubernetes. Lưu ý: Bộ nhớ không heap bao gồm bộ nhớ off-heap (khi spark.memory.offHeap.enabled = true) và bộ nhớ được sử dụng bởi các quy trình trình điều khiển khác (ví dụ: quy trình python đi với trình điều khiển PySpark) và bộ nhớ được sử dụng bởi các quy trình không phải trình điều khiển khác chạy trong cùng một container. Kích thước bộ nhớ tối đa của vùng chứa đến trình điều khiển đang chạy được xác định bằng tổng của spark.driver.memoryOverh	2.3.0

		ead và spark.driver.memory.	
spark.executor.resource.{resourceName}.amount	0	Số lượng của một loại tài nguyên cụ thể để sử dụng cho mỗi quy trình của trình thực thi. Nếu điều này được sử dụng, bạn cũng phải chỉ định spark.executor.resource.{ResourceName} .discoveryScript để trình thực thi tìm tài nguyên khi khởi động.	3.0.0
spark.executor.resource.{resourceName}.discoveryScript	None	Một tập lệnh để trình thực thi chạy để khám phá một loại tài nguyên cụ thể. Điều này sẽ ghi vào STDOUT một chuỗi JSON ở định dạng của lớp ResourceInformation. Điều này có một tên và một mảng địa chỉ.	3.0.0
spark.executor.resource.{resourceName}.vendor	None	Nhà cung cấp tài nguyên để sử dụng cho những người thực thi. Tùy chọn này hiện chỉ được hỗ trợ trên Kubernetes và thực tế là cả nhà cung cấp và miền tuân theo quy ước đặt tên plugin thiết bị Kubernetes. (ví dụ: Đối với GPU trên Kubernetes, cấu hình này sẽ được đặt thành nvidia.com hoặc amd.com)	3.0.0
spark.extraListeners	(none)	Danh sách các lớp được phân tách bằng dấu phẩy triển khai SparkListener; khi khởi tạo SparkContext, các cá thể của các lớp này sẽ được tạo và đăng ký với bus lắng nghe của Spark. Nếu một lớp có một phương thức khởi tạo một	1.3.0

		đối số chấp nhận một SparkConf, thì phương thức khởi tạo đó sẽ được gọi; nếu không, một hàm tạo không đối số sẽ được gọi. Nếu không tìm thấy hàm tạo hợp lệ nào, quá trình tạo SparkContext sẽ không thành công với một ngoại lệ.	
spark.local.dir	/tmp	Thư mục để sử dụng cho không gian "đầu" trong Spark, bao gồm các tệp đầu ra bản đồ và RDD được lưu trữ trên đĩa. Điều này phải nằm trên một đĩa cục bộ, nhanh trong hệ thống của bạn. Nó cũng có thể là một danh sách được phân tách bằng dấu phẩy gồm nhiều thư mục trên các đĩa khác nhau. Lưu ý: Điều này sẽ bị ghi đè bởi các biến môi trường SPARK_LOCAL_DIRS (Độc lập), MESOS_SANDBOX (Mesos) hoặc LOCAL_DIRS (YARN) do người quản lý cụm thiết lập.	0.5.0
spark.logConf	False	Ghi lại SparkConf hiệu quả dưới dạng INFO khi SparkContext được khởi động.	0.9.0
spark.master	(none)	Người quản lý cụm để kết nối. Xem danh sách URL chính được phép.	0.9.0
spark.submit.deployMode	(none)	Chế độ triển khai của chương trình trình điều khiển Spark, "máy khách" hoặc "cụm", có nghĩa là khởi chạy chương trình	1.5.0

		trình điều khiển cục bộ ("máy khách") hoặc từ xa ("cụm") trên một trong các nút bên trong cụm.	
spark.log.callerContext	(none)	Thông tin ứng dụng sẽ được ghi vào nhật ký Yarn RM / nhật ký kiểm tra HDFS khi chạy trên Yarn / HDFS. Độ dài của nó phụ thuộc vào cấu hình Hadoop hadoop.caller.context.max.size. Nó phải ngắn gọn và thường có thể có tối đa 50 ký tự.	2.2.0
spark.driver.supervise	False	Nếu đúng, hãy tự động khởi động lại trình điều khiển nếu nó không thành công với trạng thái thoát khác không. Chỉ có hiệu lực trong chế độ độc lập Spark hoặc chế độ triển khai cụm Mesos.	1.3.0
spark.driver.log.dfsDir	(none)	Thư mục cơ sở chứa nhật ký trình điều khiển Spark được đồng bộ hóa, nếu spark.driver.log.persistToDfs.enabled là true. Trong thư mục cơ sở này, mỗi ứng dụng ghi nhật ký trình điều khiển vào một tệp ứng dụng cụ thể. Người dùng có thể muốn đặt vị trí này thành một vị trí thống nhất như thư mục HDFS để các tệp nhật ký trình điều khiển có thể được duy trì để sử dụng sau này. Thư mục này sẽ cho phép mọi người dùng Spark đọc / ghi tệp và người dùng Spark History Server xóa tệp. Ngoài ra, các nhật ký	3.0.0

		cũ hơn từ thư mục này sẽ được Máy chủ Lịch sử Spark làm sạch nếu spark.history.fs.driverlog.cleaner.enabled là đúng và nếu chúng cũ hơn tuổi tối đa được định cấu hình bằng cách đặt spark.history.fs.driverlog.cleaner. maxAge.	
spark.driver.log.persistToDfs.enabled	False	Nếu đúng, ứng dụng spark đang chạy ở chế độ máy khách sẽ ghi nhật ký trình điều khiển vào kho lưu trữ liên tục, được định cấu hình trong spark.driver.log.dfsDir. Nếu spark.driver.log.dfsDir không được định cấu hình, nhật ký trình điều khiển sẽ không được lưu giữ. Ngoài ra, hãy bật trình dọn dẹp bằng cách đặt spark.history.fs.driverlog.cleaner.enabled thành true trong Máy chủ Lịch sử Spark.	3.0.0
spark.driver.log.layout	%d{yy/MM/dd HH:mm:ss.SSS} %t %p %c{1}: %m%n	Bố cục cho nhật ký trình điều khiển được đồng bộ hóa với spark.driver.log.dfsDir. Nếu điều này không được định cấu hình, nó sẽ sử dụng bố cục cho appender đầu tiên được xác định trong log4j.properties. Nếu điều đó cũng không được định cấu hình, nhật ký trình điều khiển sử dụng bố cục mặc định.	3.0.0
spark.driver.log.allowErasureCoding	False	Có cho phép nhật ký trình điều khiển sử dụng mã xóa	3.0.0

		hay không. Trên HDFS, các tệp được mã hóa xóa sẽ không cập nhật nhanh như các tệp sao chép thông thường, do đó, chúng mất nhiều thời gian hơn để phản ánh các thay đổi do ứng dụng viết. Lưu ý rằng ngay cả khi điều này là đúng, Spark vẫn sẽ không buộc tệp sử dụng mã hóa xóa, nó sẽ chỉ sử dụng mặc định của hệ thống tệp	

Runtime Environment

Property Name	Default	Meaning	Since Version
spark.driver.extraClassPath	(none)	Các mục nhập classpath bổ sung để thêm trước vào classpath của trình điều khiển. Lưu ý: Ở chế độ máy khách, cấu hình này không được đặt thông qua SparkConf trực tiếp trong ứng dụng của bạn, vì trình điều khiển JVM đã bắt đầu tại thời điểm đó. Thay vào đó, vui lòng đặt điều này thông qua tùy chọn dòng lệnh --driver-class-path hoặc trong tệp thuộc tính mặc định của bạn.	1.0.0
spark.driver.defaultJavaOptions	(none)	Một chuỗi các tùy chọn JVM mặc định để thêm vào spark.driver.extraJavaOptions. Điều này được thiết lập bởi các quản trị viên. Ví dụ: cài đặt GC hoặc ghi nhật ký khác. Lưu ý rằng việc đặt cài đặt kích thước heap tối đa (-Xmx) với tùy chọn này là	3.0.0

		bất hợp pháp. Có thể đặt cài đặt kích thước đống tối đa bằng spark.driver.memory ở chế độ cụm và thông qua tùy chọn dòng lệnh --driver-memory trong chế độ máy khách. Lưu ý: Ở chế độ máy khách, cấu hình này không được đặt thông qua SparkConf trực tiếp trong ứng dụng của bạn, vì trình điều khiển JVM đã bắt đầu tại thời điểm đó. Thay vào đó, hãy đặt điều này thông qua tùy chọn dòng lệnh --driver-java-options hoặc trong tệp thuộc tính mặc định của bạn.	
spark.driver.extraJavaOptions	(none)	Một chuỗi các tùy chọn JVM bổ sung để chuyển cho người lái xe. Điều này là do người dùng thiết lập. Ví dụ: cài đặt GC hoặc ghi nhật ký khác. Lưu ý rằng việc đặt cài đặt kích thước heap tối đa (-Xmx) với tùy chọn này là bất hợp pháp. Có thể đặt cài đặt kích thước đống tối đa bằng spark.driver.memory ở chế độ cụm và thông qua tùy chọn dòng lệnh --driver-memory trong chế độ máy khách. Lưu ý: Ở chế độ máy khách, cấu hình này không được đặt thông qua SparkConf trực tiếp trong ứng dụng của bạn, vì trình điều khiển JVM đã bắt đầu tại thời điểm đó. Thay vào đó, hãy đặt điều này thông qua tùy chọn dòng lệnh --driver-java-options hoặc trong tệp thuộc tính	1.0.0

		mặc định của bạn. spark.driver.defaultJavaOptions sẽ được thêm vào cấu hình này.	
spark.driver.extraLibraryPath	(none)	Đặt một đường dẫn thư viện đặc biệt để sử dụng khi khởi chạy trình điều khiển JVM. Lưu ý: Ở chế độ máy khách, cấu hình này không được đặt thông qua SparkConf trực tiếp trong ứng dụng của bạn, vì trình điều khiển JVM đã bắt đầu tại thời điểm đó. Thay vào đó, vui lòng đặt điều này thông qua tùy chọn dòng lệnh --driver-library-path hoặc trong tệp thuộc tính mặc định của bạn.	1.0.0
spark.driver.userClassPathFirst	False	(Thử nghiệm) Có ưu tiên các lọ do người dùng thêm vào các lọ của chính Spark khi tải các lớp trong trình điều khiển hay không. Tính năng này có thể được sử dụng để giảm thiểu xung đột giữa phụ thuộc của Spark và phụ thuộc của người dùng. Nó hiện là một tính năng thử nghiệm. Điều này chỉ được sử dụng trong chế độ cụm.	1.3.0
spark.executor.extraClassPath	(none)	Các mục nhập classpath bổ sung để thêm trước vào classpath của những người thực thi. Điều này tồn tại chủ yếu để tương thích ngược với các phiên bản Spark cũ hơn. Người dùng thường không cần đặt tùy chọn này.	1.0.0
spark.executor.defaultJavaOptions	(none)	Một chuỗi các tùy chọn JVM mặc định để thêm trước vào spark.executor.extraJavaOpt	3.0.0

		ions. Điều này được thiết lập bởi các quản trị viên. Ví dụ: cài đặt GC hoặc ghi nhật ký khác. Lưu ý rằng việc đặt thuộc tính Spark hoặc cài đặt kích thước heap tối đa (-Xmx) với tùy chọn này là bất hợp pháp. Thuộc tính Spark nên được đặt bằng đối tượng SparkConf hoặc tệp spark-defaults.conf được sử dụng với tập lệnh spark-submit. Có thể đặt cài đặt kích thước đống tối đa bằng spark.executor.memory. Các ký hiệu sau, nếu có sẽ được nội suy: sẽ được thay thế bằng ID ứng dụng và sẽ được thay thế bằng ID người thực thi. Ví dụ: để bật tính năng ghi nhật ký gc chi tiết vào tệp được đặt tên cho ID thực thi của ứng dụng trong / tmp, hãy chuyển 'giá trị' là: -verbose: gc -Xloggc: / tmp / -. Gc	
spark.executor.extraJavaOptions	(none)	Một chuỗi các tùy chọn JVM bổ sung để chuyển cho người thực thi. Điều này là do người dùng thiết lập. Ví dụ: cài đặt GC hoặc ghi nhật ký khác. Lưu ý rằng việc đặt thuộc tính Spark hoặc cài đặt kích thước heap tối đa (-Xmx) với tùy chọn này là bất hợp pháp. Thuộc tính Spark nên được đặt bằng đối tượng SparkConf hoặc tệp spark-defaults.conf được sử dụng với tập lệnh spark-submit. Có thể đặt cài đặt kích thước đống tối đa bằng spark.executor.memory. Các	1.0.0

		ký hiệu sau, nếu có sẽ được nội suy: sẽ được thay thế bằng ID ứng dụng và sẽ được thay thế bằng ID người thực thi. Ví dụ: để bật tính năng ghi nhật ký gc chi tiết vào tệp được đặt tên cho ID thực thi của ứng dụng trong / tmp, hãy chuyển 'giá trị' của: -verbose: gc -Xloggc: / tmp / -. Gc spark.executor.defaultJavaOptions sẽ là thêm vào cấu hình này.	
spark.executor.extraLibraryPath	(none)	Đặt một đường dẫn thư viện đặc biệt để sử dụng khi khởi chạy JVM của trình thực thi.	1.0.0
spark.executor.logs.rolling.maxRetainedFiles	(none)	Đặt số lượng tệp nhật ký luân phiên mới nhất sẽ được hệ thống giữ lại. Các tệp nhật ký cũ hơn sẽ bị xóa. Bị tắt theo mặc định.	1.1.0
spark.executor.logs.rolling.enableCompression	False	Bật nén nhật ký thực thi. Nếu nó được kích hoạt, nhật ký trình thực thi được cuộn sẽ được nén. Bị tắt theo mặc định.	2.0.2
spark.executor.logs.rolling.maxSize	(none)	Đặt chiến lược cuộn nhật ký của người thực thi. Theo mặc định, nó bị tắt. Nó có thể được đặt thành "time" (lăn theo thời gian) hoặc "size" (lăn theo kích thước). Đối với "thời gian", hãy sử dụng spark.executor.logs.rolling.time.interval để đặt khoảng thời gian luân phiên. Đối với "kích thước", sử dụng spark.executor.logs.rolling.maxSize để đặt kích thước tệp tối đa cho cuộn.	1.1.0

spark.executor.logs.rolling.time.interval	Daily	Đặt khoảng thời gian mà nhật ký trình thực thi sẽ được cuộn qua. Tính năng cuộn bị tắt theo mặc định. Các giá trị hợp lệ là hàng ngày, hàng giờ, hàng phút hoặc bất kỳ khoảng thời gian nào tính bằng giây. Xem spark.executor.logs.rolling. maxRetainedFiles để biết tự động làm sạch nhật ký cũ.	1.1.0
spark.executor.userClassPathFirst	False	(Thử nghiệm) Chức năng tương tự như spark.driver.userClassPathFirst, nhưng được áp dụng cho các phiên bản thực thi.	1.3.0
spark.executorEnv.[EnvironmentVariableName]	(none)	Thêm biến môi trường được chỉ định bởi EnvironmentVariableName vào quy trình Executor. Người dùng có thể chỉ định nhiều trong số này để đặt nhiều biến môi trường.	0.9.0
spark.redaction.regex	(?i)secret|password|token	Regex để quyết định các thuộc tính cấu hình Spark và biến môi trường nào trong môi trường trình điều khiển và trình thực thi chứa thông tin nhạy cảm. Khi regex này khớp với khóa hoặc giá trị thuộc tính, giá trị sẽ được biên dịch lại từ giao diện người dùng môi trường và các nhật ký khác nhau như YARN và nhật ký sự kiện.	2.1.2
spark.python.profile	False	Bật cấu hình trong Python worker, kết quả cấu hình sẽ hiển thị bằng sc.show_profiles () hoặc nó sẽ được hiển thị trước khi trình điều khiển thoát. Nó cũng có thể được đưa vào đĩa bằng sc.dump_profiles	1.2.0

		(đường dẫn). Nếu một số kết quả hồ sơ đã được hiển thị theo cách thủ công, chúng sẽ không được hiển thị tự động trước khi người lái xe thoát ra. Theo mặc định, pyspark.profiler.BasicProfiler sẽ được sử dụng, nhưng điều này có thể bị ghi đè bằng cách chuyển một lớp hồ sơ vào làm tham số cho hàm tạo SparkContext.	
spark.python.profile.dump	(none)	Thư mục được sử dụng để kết xuất kết quả hồ sơ trước khi trình điều khiển thoát. Kết quả sẽ được kết xuất dưới dạng tệp riêng biệt cho từng RDD. Chúng có thể được tải bởi pstats.Stats (). Nếu điều này được chỉ định, kết quả hồ sơ sẽ không được hiển thị tự động.	1.2.0
spark.python.worker.memory	512m	Dung lượng bộ nhớ sẽ sử dụng cho mỗi quá trình python worker trong quá trình tổng hợp, ở định dạng giống như chuỗi bộ nhớ JVM với hậu tố đơn vị kích thước ("k", "m", "g" hoặc "t") (ví dụ: 512m, 2g). Nếu bộ nhớ được sử dụng trong quá trình tổng hợp vượt quá dung lượng này, nó sẽ tràn dữ liệu vào đĩa.	1.1.0
spark.python.worker.reuse	True	Sử dụng lại công nhân Python hay không. Nếu có, nó sẽ sử dụng một số lượng nhân viên Python cố định, không cần fork () một quy trình Python cho mọi tác vụ. Nó sẽ rất hữu ích nếu có một chương trình phát sóng lớn, khi đó chương trình phát	1.2.0

		sóng sẽ không cần phải chuyển từ JVM sang Python worker cho mọi tác vụ.	
spark.files		Danh sách tệp được phân tách bằng dấu phẩy sẽ được đặt trong thư mục làm việc của mỗi trình thực thi. Quả cầu được cho phép.	1.0.0
spark.submit.pyFiles		Danh sách các tệp .zip, .egg hoặc .py được phân tách bằng dấu phẩy để đặt trên ứng dụng PYTHONPATH cho Python. Quả cầu được cho phép.	1.0.1
spark.jars		Danh sách các lọ được phân tách bằng dấu phẩy để đưa vào đường dẫn trình điều khiển và trình thực thi. Quả cầu được cho phép.	0.9.0
spark.jars.packages		Danh sách được phân tách bằng dấu phẩy gồm các tọa độ Maven của các chum để đưa vào các đường dẫn của trình điều khiển và trình thực thi. Các tọa độ phải là groupId: artistId: version. Nếu spark.jars.ivySettings được cung cấp, phần mềm sẽ được giải quyết theo cấu hình trong tệp, nếu không, phần tạo tác sẽ được tìm kiếm trong kho maven cục bộ, sau đó là trung tâm maven và cuối cùng là bất kỳ kho lưu trữ từ xa bổ sung nào được cung cấp bởi tùy chọn dòng lệnh - kho lưu trữ. Để biết thêm chi tiết, hãy xem Quản lý phụ thuộc nâng cao.	1.5.0
spark.jars.excludes		Danh sách được phân tách bằng dấu phẩy của groupId:	1.5.0

		craftId, để loại trừ trong khi giải quyết các phần phụ thuộc được cung cấp trong spark.jars.packages để tránh xung đột phụ thuộc.	
spark.jars.ivy		Đường dẫn để chỉ định thư mục người dùng Ivy, được sử dụng cho bộ nhớ cache Ivy cục bộ và các tệp gói từ spark.jars.packages. Điều này sẽ ghi đè thuộc tính Ivy ivy.default.ivy.user.dir được mặc định thành ~ / .ivy2.	1.3.0
spark.jars.ivySettings		Đường dẫn đến tệp cài đặt Ivy để tùy chỉnh độ phân giải của các lọ được chỉ định bằng spark.jars.packages thay vì các cài đặt mặc định tích hợp, chẳng hạn như maven center. Các kho lưu trữ bổ sung được cung cấp bởi tùy chọn dòng lệnh --repositories hoặc spark.jars.repositories cũng sẽ được bao gồm. Hữu ích khi cho phép Spark giải quyết các hiện vật từ phía sau tường lửa, ví dụ: thông qua một máy chủ tạo tác nội bộ như Artifactory. Bạn có thể tìm thấy chi tiết về định dạng tệp cài đặt tại Tệp Cài đặt	2.2.0
spark.jars.repositories		Danh sách các kho lưu trữ từ xa bổ sung được phân tách bằng dấu phẩy để tìm kiếm các tọa độ maven được cung cấp bằng --packages hoặc spark.jars.packages.	2.3.0
spark.pyspark.driver.python		Thực thi nhị phân Python để sử dụng cho PySpark trong trình điều khiển. (mặc định là spark.pyspark.python)	2.1.0

spark.pyspark.python		Thực thi nhị phân Python để sử dụng cho PySpark trong cả trình điều khiển và trình thực thi.	2.1.0

Shuffle Behavior

Property Name	Default	Meaning	Since Version
spark.reducer.maxSizeInFlight	48m	Kích thước tối đa của đầu ra bản đồ để tìm nạp đồng thời từ mỗi tác vụ giảm, trong MiB trừ khi được chỉ định khác. Vì mỗi đầu ra yêu cầu chúng ta tạo một bộ đệm để nhận nó, điều này đại diện cho chi phí bộ nhớ cố định cho mỗi tác vụ giảm, vì vậy hãy giữ nó nhỏ trừ khi bạn có một lượng lớn bộ nhớ.	1.4.0
spark.reducer.maxReqsInFlight	Int.MaxValue	Cấu hình này giới hạn số lượng yêu cầu từ xa để tìm nạp các khối tại bất kỳ điểm nhất định nào. Khi số lượng máy chủ trong cụm tăng lên, nó có thể dẫn đến số lượng rất lớn các kết nối gửi đến một hoặc nhiều nút, khiến các công nhân bị lỗi khi tải. Bằng cách cho phép nó giới hạn số lượng yêu cầu tìm nạp, tình huống này có thể được giảm thiểu.	2.0.0
spark.reducer.maxBlocksInFlightPerAddress	Int.MaxValue	Cấu hình này giới hạn số lượng khối từ xa được tìm nạp cho mỗi tác vụ giảm từ một cổng máy chủ nhất định. Khi một số lượng lớn các khối	2.2.1

		đang được yêu cầu từ một địa chỉ nhất định trong một lần tìm nạp hoặc đồng thời, điều này có thể làm hỏng trình thực thi phục vụ hoặc Trình quản lý nút. Điều này đặc biệt hữu ích để giảm tải trên Node Manager khi bật chế độ trộn bên ngoài. Bạn có thể giảm thiểu vấn đề này bằng cách đặt nó thành một giá trị thấp hơn.	
spark.shuffle.compress	True	Có nén các tệp đầu ra bản đồ hay không. Nói chung là một ý kiến hay. Nén sẽ sử dụng spark.io.compression.codec.	0.6.0
spark.shuffle.file.buffer	32k	Kích thước của bộ đệm trong bộ nhớ cho mỗi luồng đầu ra tệp trộn, trong KiB trừ khi được chỉ định khác. Các bộ đệm này làm giảm số lần tìm đĩa và các lệnh gọi hệ thống được thực hiện trong việc tạo các tệp ngẫu nhiên trung gian.	1.4.0
spark.shuffle.io.maxRetries	3	(Chỉ Netty) Các lần tìm nạp không thành công do các ngoại lệ liên quan đến IO sẽ tự động được thử lại nếu giá trị này được đặt thành giá trị khác 0. Logic thử lại này giúp ổn định các xáo trộn lớn khi đối mặt với các lần tạm dừng GC kéo dài hoặc các sự cố kết nối mạng tạm thời.	1.2.0

spark.shuffle.io.numConnectionsPerPeer	1	(Chỉ mạng) Kết nối giữa các máy chủ được sử dụng lại để giảm tích tụ kết nối cho các cụm lớn. Đối với các cụm có nhiều đĩa cứng và ít máy chủ, điều này có thể dẫn đến không đủ đồng thời để bão hòa tất cả các đĩa và do đó người dùng có thể cân nhắc việc tăng giá trị này.	1.2.1
spark.shuffle.io.preferDirectBufs	True	(Chỉ Netty) Bộ đệm off-heap được sử dụng để giảm việc thu gom rác trong quá trình trộn và chuyển khối bộ nhớ cache. Đối với các môi trường mà bộ nhớ off-heap bị giới hạn chặt chẽ, người dùng có thể muốn tắt tính năng này để buộc tất cả các phân bổ từ Netty ở chế độ on-heap.	1.2.0
spark.shuffle.io.retryWait	5s	(Chỉ mạng) Thời gian chờ giữa các lần tìm nạp lại. Độ trễ tối đa do thử lại là 15 giây theo mặc định, được tính bằng maxRetries * retryWait.	1.2.1
spark.shuffle.io.backLog	-1	Độ dài của hàng đợi chấp nhận cho dịch vụ trộn. Đối với các ứng dụng lớn, giá trị này có thể cần được tăng lên để các kết nối đến không bị giảm nếu dịch vụ không thể theo kịp với số lượng lớn các kết nối đến trong một khoảng thời gian ngắn. Điều này cần được định cấu hình ở bất kỳ	1.1.1

		nơi nào mà bản thân dịch vụ trộn đang chạy, có thể nằm ngoài ứng dụng (xem tùy chọn spark.shuffle.service.enabled bên dưới). Nếu được đặt dưới 1, sẽ dự phòng về mặc định của hệ điều hành được xác định bởi io.netty.util.NetUtil # SOMAXCONN của Netty.	
spark.shuffle.service.enabled	false	Bật dịch vụ trộn bên ngoài. Dịch vụ này lưu giữ các tệp xáo trộn được viết bởi những người thực thi để những người thực thi có thể được xóa một cách an toàn. Điều này phải được bật nếu spark.dynamicAllocation.enabled là "true". Dịch vụ trộn bên ngoài phải được thiết lập để kích hoạt nó. Xem cấu hình phân bổ động và tài liệu thiết lập để biết thêm thông tin.	1.2.0
spark.shuffle.service.enabled	False	Bật dịch vụ trộn bên ngoài. Dịch vụ này lưu giữ các tệp xáo trộn được viết bởi những người thực thi để những người thực thi có thể được xóa một cách an toàn. Điều này phải được bật nếu spark.dynamicAllocation.enabled là "true". Dịch vụ trộn bên ngoài phải được thiết lập để kích	1.2.0

		hoạt nó. Xem cấu hình phân bổ động và tài liệu thiết lập để biết thêm thông tin.	
spark.shuffle.service.port	7337	Cổng mà dịch vụ trộn bên ngoài sẽ chạy.	1.2.0
spark.shuffle.service.index.cache.size	100m	Các mục nhập trong bộ nhớ cache được giới hạn trong vùng nhớ được chỉ định, tính bằng byte trừ khi được chỉ định khác.	2.3.0
spark.shuffle.maxChunksBeingTransferred	Long.MAX_VALUE	Số lượng tối đa các khối được phép chuyển cùng một lúc trên dịch vụ xáo trộn. Lưu ý rằng các kết nối mới sẽ bị đóng khi đạt đến số lượng tối đa. Máy khách sẽ thử lại theo cấu hình trộn lại thử (xem spark.shuffle.io.maxRetries và spark.shuffle.io.retryWait), nếu đạt đến các giới hạn đó, tác vụ sẽ không thành công với lỗi tìm nạp.	2.3.0
spark.shuffle.sort.bypassMergeThreshold	200	(Nâng cao) Trong trình quản lý xáo trộn dựa trên sắp xếp, tránh sắp xếp dữ liệu hợp nhất nếu không có tổng hợp phía bản đồ và có nhiều nhất là nhiều phân vùng giảm.	1.1.1
spark.shuffle.spill.compress	True	Có nén dữ liệu bị tràn trong quá trình xáo trộn hay không. Nén sẽ sử dụng spark.io.compression.codec.	0.9.0
spark.shuffle.accurateBlockThreshold	100 * 1024 * 1024	Ngưỡng tính bằng byte trên đó kích thước của khối trộn trong	2.2.1

		HighlyCompressedMapStatus được ghi lại chính xác. Điều này giúp ngăn chặn OOM bằng cách tránh đánh giá thấp kích thước khối trộn khi tìm nạp khối trộn.	
spark.shuffle.registration.timeout	5000	Thời gian chờ tính bằng mili giây để đăng ký dịch vụ trộn bên ngoài.	2.3.0
spark.shuffle.registration.maxAttempts	3	Khi chúng tôi không đăng ký được với dịch vụ xáo trộn bên ngoài, chúng tôi sẽ thử lại trong thời gian maxAttempts.	2.3.0

Spark UI

Property Name	Default	Meaning	Since Version
spark.eventLog.logBlockUpdates.enabled	False	Có ghi lại các sự kiện cho mọi bản cập nhật khối hay không, nếu spark.eventLog.enabled là true. * Cảnh báo *: Điều này sẽ làm tăng đáng kể kích thước của nhật ký sự kiện.	2.3.0
spark.eventLog.longForm.enabled	False	Nếu đúng, hãy sử dụng biểu mẫu dài của các trang web cuộc gọi trong nhật ký sự kiện. Nếu không, hãy sử dụng mẫu ngắn.	2.4.0
spark.eventLog.compress	False	Có nén các sự kiện đã ghi hay không, nếu spark.eventLog.enabled là true.	1.0.0
spark.eventLog.compression.codec		Codec để nén các sự kiện đã ghi. Nếu điều này không được cung cấp, spark.io.compression.codec sẽ được sử dụng.	3.0.0

spark.eventLog.erasureCoding.enabled	False	Cho phép nhật ký sự kiện sử dụng mã hóa xóa hay tắt mã hóa xóa, bất kể giá trị mặc định của hệ thống tệp. Trên HDFS, các tệp được mã hóa xóa sẽ không cập nhật nhanh như các tệp sao chép thông thường, do đó, các bản cập nhật ứng dụng sẽ mất nhiều thời gian hơn để xuất hiện trong Máy chủ Lịch sử. Lưu ý rằng ngay cả khi điều này là đúng, Spark vẫn sẽ không buộc tệp sử dụng mã hóa xóa, nó sẽ chỉ sử dụng mặc định của hệ thống tệp.	3.0.0
spark.eventLog.dir	file:///tmp/spark-events	Thư mục cơ sở ghi lại các sự kiện Spark, nếu spark.eventLog.enabled là true. Trong thư mục cơ sở này, Spark tạo một thư mục con cho mỗi ứng dụng và ghi nhật ký các sự kiện cụ thể cho ứng dụng trong thư mục này. Người dùng có thể muốn đặt vị trí này thành một vị trí thống nhất như thư mục HDFS để máy chủ lịch sử có thể đọc các tệp lịch sử.	1.0.0
spark.eventLog.enabled	False	Có ghi lại các sự kiện Spark hay không, hữu ích cho việc tạo lại giao diện người dùng Web sau khi ứng dụng hoàn tất.	1.0.0
spark.eventLog.overwrite	False	Có ghi đè lên bất kỳ tệp hiện có nào không.	1.0.0
spark.eventLog.buffer.kb	100k	Kích thước bộ đệm để sử dụng khi ghi vào các luồng đầu ra, trong KiB trừ khi được chỉ định khác.	1.0.0

spark.eventLog.rolling.enabled	False	Việc cuộn qua các tệp nhật ký sự kiện có được bật hay không. Nếu được đặt thành true, nó sẽ cắt từng tệp nhật ký sự kiện xuống kích thước đã định cấu hình.	3.0.0
spark.eventLog.rolling.maxFileSize	128m	Khi spark.eventLog.rolling.enabled = true, chỉ định kích thước tối đa của tệp nhật ký sự kiện trước khi cuộn qua.	3.0.0
spark.ui.dagGraph.retainedRootRDDs	Int.MaxValue	Có bao nhiêu nút đồ thị DAG mà giao diện người dùng Spark và API trạng thái nhớ trước khi thu gom rác.	2.1.0
spark.ui.enabled	true	Có chạy giao diện người dùng web cho ứng dụng Spark hay không.	1.1.1
spark.ui.killEnabled	true	Cho phép loại bỏ các công việc và giai đoạn khỏi giao diện người dùng web.	1.0.0
spark.ui.liveUpdate.period	100ms	Tần suất cập nhật các thực thể trực tiếp. -1 có nghĩa là "không bao giờ cập nhật" khi phát lại ứng dụng, nghĩa là chỉ lần ghi cuối cùng sẽ xảy ra. Đối với các ứng dụng trực tiếp, điều này tránh một số thao tác mà chúng ta có thể sống mà không có khi xử lý nhanh các sự kiện tác vụ đến.	2.3.0
spark.ui.liveUpdate.minFlushPeriod	1s	Thời gian tối thiểu trôi qua trước khi dữ liệu UI cũ được xóa. Điều này giúp tránh sự trì trệ của giao diện người dùng khi các sự kiện tác vụ đến không được kích hoạt thường xuyên.	2.4.2
spark.ui.port	4040	Cổng cho bảng điều khiển của ứng dụng của bạn, nơi	0.7.0

		hiển thị dữ liệu bộ nhớ và khối lượng công việc.	
spark.ui.retainedJobs	1000	Giao diện người dùng Spark và API trạng thái ghi nhớ bao nhiêu công việc trước khi thu gom rác. Đây là mức tối đa mục tiêu và có thể giữ lại ít phần tử hơn trong một số trường hợp.	1.2.0
spark.ui.retainedStages	1000	Giao diện người dùng Spark và API trạng thái ghi nhớ bao nhiêu giai đoạn trước khi thu gom rác. Đây là mức tối đa mục tiêu và có thể giữ lại ít phần tử hơn trong một số trường hợp.	0.9.0
spark.ui.retainedTasks	100000	Có bao nhiêu tác vụ trong một giai đoạn mà giao diện người dùng Spark và API trạng thái ghi nhớ trước khi thu gom rác. Đây là mức tối đa mục tiêu và có thể giữ lại ít phần tử hơn trong một số trường hợp.	2.0.1
spark.ui.reverseProxy	False	Cho phép chạy Spark Master làm proxy ngược cho giao diện người dùng ứng dụng và công nhân. Trong chế độ này, Spark master sẽ proxy ngược lại giao diện người dùng của worker và ứng dụng để cho phép truy cập mà không yêu cầu quyền truy cập trực tiếp vào máy chủ của chúng. Hãy sử dụng nó một cách thận trọng vì giao diện người dùng của worker và ứng dụng sẽ không thể truy cập trực tiếp được, bạn sẽ chỉ có thể truy cập chúng thông qua URL công khai của spark master / proxy.	2.1.0

		Cài đặt này ảnh hưởng đến tất cả công nhân và giao diện người dùng ứng dụng đang chạy trong cụm và phải được đặt trên tất cả công nhân, trình điều khiển và chủ.	
spark.ui.reverseProxyUrl		Đây là URL nơi proxy của bạn đang chạy. URL này dành cho proxy đang chạy trước Spark Master. Điều này hữu ích khi chạy proxy để xác thực, ví dụ: Proxy OAuth. Đảm bảo rằng đây là một URL hoàn chỉnh bao gồm lược đồ (http / https) và cổng để truy cập proxy của bạn.	2.1.0
spark.ui.proxyRedirectUri		Nơi giải quyết các chuyển hướng khi Spark đang chạy sau proxy. Điều này sẽ làm cho Spark sửa đổi các phản hồi chuyển hướng để chúng trỏ đến máy chủ proxy, thay vì địa chỉ riêng của Spark UI. Đây chỉ nên là địa chỉ của máy chủ, không có bất kỳ đường dẫn tiền tố nào cho ứng dụng; tiền tố phải được đặt bởi chính máy chủ proxy (bằng cách thêm tiêu đề yêu cầu X-Forwarded-Context) hoặc bằng cách đặt cơ sở proxy trong cấu hình của ứng dụng Spark.	3.0.0
spark.ui.showConsoleProgress	False	Hiển thị thanh tiến trình trong bảng điều khiển. Thanh tiến trình hiển thị tiến trình của các giai đoạn chạy lâu hơn 500 mili giây. Nếu nhiều giai đoạn chạy cùng một lúc, nhiều thanh	1.2.1

		tiến trình sẽ được hiển thị trên cùng một dòng. Lưu ý: Trong môi trường shell, giá trị mặc định của spark.ui.showConsoleProgress là true.	
spark.ui.custom.executor.log.url	(none)	Chỉ định URL nhật ký trình thực thi tia lửa tùy chỉnh để hỗ trợ dịch vụ nhật ký bên ngoài thay vì sử dụng URL nhật ký ứng dụng của trình quản lý cụm trong giao diện người dùng Spark. Spark sẽ hỗ trợ một số biến đường dẫn thông qua các mẫu có thể khác nhau trên trình quản lý cụm. Vui lòng kiểm tra tài liệu dành cho người quản lý cụm của bạn để xem các mẫu nào được hỗ trợ, nếu có. Xin lưu ý rằng cấu hình này cũng thay thế các url nhật ký gốc trong nhật ký sự kiện, điều này cũng sẽ có hiệu lực khi truy cập ứng dụng trên máy chủ lịch sử. Các url nhật ký mới phải là vĩnh viễn, nếu không, bạn có thể có liên kết chết cho các url nhật ký của người thực thi. Hiện tại, chỉ có chế độ YARN hỗ trợ cấu hình này	3.0.0
spark.worker.ui.retainedExecutors	1000	Có bao nhiêu trình thực thi hoàn thành mà giao diện người dùng Spark và API trạng thái nhớ trước khi thu gom rác.	1.5.0
spark.worker.ui.retainedDrivers	1000	Có bao nhiêu trình điều khiển hoàn thành mà giao diện người dùng Spark và	1.5.0

		API trạng thái nhớ trước khi thu gom rác.	
spark.sql.ui.retainedExecutions	1000	Số lần thực thi hoàn thành mà giao diện người dùng Spark và API trạng thái ghi nhớ trước khi thu gom rác.	1.5.0
spark.streaming.ui.retainedBatches	1000	Có bao nhiêu lô hoàn thành mà giao diện người dùng Spark và API trạng thái nhớ trước khi thu gom rác.	1.0.0
spark.ui.retainedDeadExecutors	100	Có bao nhiêu người thực thi đã chết mà giao diện người dùng Spark và API trạng thái nhớ trước khi thu gom rác.	2.0.0
spark.ui.filters	None	Danh sách tên lớp bộ lọc được phân tách bằng dấu phẩy để áp dụng cho giao diện người dùng Web Spark. Bộ lọc phải là Bộ lọc servlet javax tiêu chuẩn. Các thông số bộ lọc cũng có thể được chỉ định trong cấu hình, bằng cách thiết lập các mục cấu hình của biểu mẫu spark. <Tên lớp của bộ lọc> .param. <Tên tham số> = <giá trị> Ví dụ: spark.ui.filters = com.test.filter1 spark.com.test.filter1.param.name1 = foo spark.com.test.filter1.param.name2 = bar	1.0.0
spark.ui.requestHeaderSize	8k	Kích thước tối đa được phép cho tiêu đề yêu cầu HTTP, tính bằng byte trừ khi được chỉ định khác. Cài đặt này cũng áp dụng cho Máy chủ Lịch sử Spark.	2.2.3

Compression and Serialization

- spark.broadcast.compress
- spark.checkpoint.compress
- spark.io.compression.codec
- spark.io.compression.lz4.blockSize
- spark.io.compression.snappy.blockSize
- spark.io.compression.zstd.level
- spark.io.compression.zstd.bufferSize
- spark.kryo.classesToRegister
- spark.kryo.referenceTracking
- spark.kryo.registrationRequired
- spark.kryo.registrator
- spark.kryo.unsafe
- spark.kryoserializer.buffer.max
- spark.kryoserializer.buffer
- spark.rdd.compress
- spark.serializer
- spark.serializer.objectStreamReset

Memory Management

- spark.memory.fraction
- spark.memory.storageFraction
- spark.memory.offHeap.enabled
- spark.memory.offHeap.size
- spark.storage.replication.proactive
- spark.cleaner.periodicGC.interval
- spark.cleaner.referenceTracking
- spark.cleaner.referenceTracking.blocking
- spark.cleaner.referenceTracking.blocking.shuffle
- spark.cleaner.referenceTracking.cleanCheckpoints

Excution Behavior

- spark.broadcast.blockSize
- spark.broadcast.checksum
- spark.executor.cores
- spark.default.parallelism
- spark.executor.heartbeatInterval
- spark.files.fetchTimeout
- spark.files.useFetchCache
- spark.files.overwrite
- spark.files.maxPartitionBytes
- spark.files.openCostInBytes
- spark.hadoop.cloneConf
- spark.hadoop.validateOutputSpecs
- spark.storage.memoryMapThreshold
- spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version

Excution Metrics

- spark.eventLog.logStageExecutorMetrics
- spark.executor.processTreeMetrics.enabled
- spark.executor.metrics.pollingInterval

Networking

- spark.rpc.message.maxSize
- spark.blockManager.port
- spark.driver.blockManager.port
- spark.driver.bindAddress
- spark.driver.host
- spark.driver.port
- spark.rpc.io.backLog

- spark.network.timeout
- spark.network.io.preferDirectBufs
- spark.port.maxRetries
- spark.rpc.numRetries
- spark.rpc.retry.wait
- spark.rpc.askTimeout
- spark.rpc.lookupTimeout
- spark.core.connection.ack.wait.timeout
- spark.network.maxRemoteBlockSizeFetchToMem

Scheduling

- spark.cores.max
- spark.locality.wait
- spark.locality.wait.node
- spark.locality.wait.process
- spark.locality.wait.rack
- spark.scheduler.maxRegisteredResourcesWaitingTime
- spark.scheduler.minRegisteredResourcesRatio
- spark.scheduler.mode
- spark.scheduler.revive.interval
- spark.scheduler.listenerbus.eventqueue.capacity
- spark.scheduler.listenerbus.eventqueue.shared.capacity
- spark.scheduler.listenerbus.eventqueue.appStatus.capacity
- spark.scheduler.listenerbus.eventqueue.executorManagement.capacity
- spark.scheduler.listenerbus.eventqueue.eventLog.capacity
- spark.scheduler.listenerbus.eventqueue.streams.capacity
- spark.scheduler.blacklist.unschedulableTaskSetTimeout
- spark.blacklist.enabled
- spark.blacklist.timeout

- spark.blacklist.task.maxTaskAttemptsPerExecutor
- spark.blacklist.task.maxTaskAttemptsPerNode
- spark.blacklist.stage.maxFailedTasksPerExecutor
- spark.blacklist.stage.maxFailedExecutorsPerNode
- spark.blacklist.application.maxFailedTasksPerExecutor
- spark.blacklist.application.maxFailedExecutorsPerNode
- spark.blacklist.killBlacklistedExecutors
- spark.blacklist.application.fetchFailure.enabled
- spark.speculation
- spark.speculation.interval
- spark.speculation.multiplier
- spark.speculation.quantile
- spark.speculation.task.duration.threshold
- spark.task.cpus
- spark.task.resource.{resourceName}.amount
- spark.task.maxFailures
- spark.task.reaper.enabled
- spark.task.reaper.pollingInterval
- spark.task.reaper.pollingInterval
- spark.task.reaper.killTimeout
- spark.stage.maxConsecutiveAttempts

Barrie Execution Mode

- spark.barrier.sync.timeout
- spark.scheduler.barrier.maxConcurrentTasksCheck.interval
- spark.scheduler.barrier.maxConcurrentTasksCheck.maxFailures

Dynamic Allocation

- spark.dynamicAllocation.enabled

- spark.dynamicAllocation.executorIdleTimeout
- spark.dynamicAllocation.cachedExecutorIdleTimeout
- spark.dynamicAllocation.initialExecutors
- spark.dynamicAllocation.maxExecutors
- spark.dynamicAllocation.minExecutors
- spark.dynamicAllocation.executorAllocationRatio
- spark.dynamicAllocation.schedulerBacklogTimeout
- spark.dynamicAllocation.sustainedSchedulerBacklogTimeout
- spark.dynamicAllocation.shuffleTracking.enabled
- spark.dynamicAllocation.shuffleTracking.timeout

Thread Configurations

Tùy thuộc vào công việc và cấu hình cụm, chúng ta có thể đặt số lượng luồng ở một số vị trí trong Spark để sử dụng hiệu quả các tài nguyên có sẵn nhằm đạt được hiệu suất tốt hơn. Trước Spark 3.0, các cấu hình luồng này áp dụng cho tất cả các vai trò của Spark, chẳng hạn như trình điều khiển, người thực thi, công nhân và chủ. Từ Spark 3.0, chúng ta có thể định cấu hình các luồng ở mức độ chi tiết tốt hơn bắt đầu từ trình điều khiển và trình thực thi. Lấy mô-đun RPC làm ví dụ trong bảng dưới đây. Đối với các mô-đun khác, chẳng hạn như xáo trộn, chỉ cần thay thế “rpc” bằng “xáo trộn” trong tên thuộc tính ngoại trừ spark. {Driver | executive} .rpc.netty.dispatcher.numThreads, chỉ dành cho mô-đun RPC.

Property Name

- spark.{driver|executor}.rpc.io.serverThreads
- spark.{driver|executor}.rpc.io.clientThreads
- spark.{driver|executor}.rpc.netty.dispatcher.numThreads

Giá trị mặc định cho số lượng khóa cấu hình liên quan đến luồng là số lõi tối thiểu được yêu cầu cho trình điều khiển hoặc trình thực thi hoặc, nếu không có giá trị đó, số lõi có sẵn cho JVM (với giới hạn trên mã cứng là số 8).

Security

Vui lòng tham khảo trang Bảo mật để biết các tùy chọn có sẵn về cách bảo mật các hệ thống con Spark khác nhau.

SparkSQL

Runtime SQL Configuration

Cấu hình SQL thời gian chạy là cấu hình Spark SQL cho mỗi phiên, có thể thay đổi. Chúng có thể được đặt với các giá trị ban đầu bằng tệp cấu hình và các tùy chọn dòng lệnh có tiền tố --conf / -c hoặc bằng cách đặt SparkConf được sử dụng để tạo SparkSession. Ngoài ra, chúng có thể được đặt và truy vấn bằng lệnh SET và đặt chúng về giá trị ban đầu bằng lệnh RESET hoặc bằng các phương thức setter và getter của SparkSession.conf trong thời gian chạy.

Property Name

- spark.sql.adaptive.advisoryPartitionSizeInBytes
- spark.sql.adaptive.coalescePartitions.enabled
- spark.sql.adaptive.coalescePartitions.initialPartitionNum
- spark.sql.adaptive.coalescePartitions.minPartitionNum
- spark.sql.adaptive.enabled
- spark.sql.adaptive.localShuffleReader.enabled
- spark.sql.adaptive.skewJoin.enabled
- spark.sql.adaptive.skewJoin.skewedPartitionFactor
- spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes
- spark.sql.ansi.enabled
- spark.sql.autoBroadcastJoinThreshold

- spark.sql.avro.compression.codec
- spark.sql.avro.deflate.level
- spark.sql.broadcastTimeout
- spark.sql.catalog.spark_catalog
- spark.sql.cbo.enabled
- spark.sql.cbo.joinReorder.dp.star.filter
- spark.sql.cbo.joinReorder.dp.threshold
- spark.sql.cbo.joinReorder.enabled
- spark.sql.cbo.planStats.enabled
- spark.sql.cbo.starSchemaDetection
- spark.sql.columnNameOfCorruptRecord
- spark.sql.csv.filterPushdown.enabled
- spark.sql.datetime.java8API.enabled
- spark.sql.debug.maxToStringFields
- spark.sql.defaultCatalog
- spark.sql.execution.arrow.enabled
- spark.sql.execution.arrow.fallback.enabled
- spark.sql.execution.arrow.maxRecordsPerBatch
- spark.sql.execution.arrow.pyspark.enabled
- spark.sql.execution.arrow.pyspark.fallback.enabled
- spark.sql.execution.arrow.sparkr.enabled
- spark.sql.execution.pandas.udf.buffer.size
- spark.sql.files.ignoreCorruptFiles
- spark.sql.files.ignoreMissingFiles
- spark.sql.files.maxPartitionBytes
- spark.sql.files.maxRecordsPerFile
- spark.sql.function.concatBinaryAsString
- spark.sql.function.eltOutputAsString
- spark.sql.groupByAliases

- spark.sql.groupByOrdinal
- spark.sql.hive.convertInsertingPartitionedTable
- spark.sql.hive.convertMetastoreCtas
- spark.sql.hive.convertMetastoreOrc
- spark.sql.hive.convertMetastoreParquet
- spark.sql.hive.convertMetastoreParquet.mergeSchema
- spark.sql.hive.filesourcePartitionFileCacheSize
- spark.sql.hive.manageFilesourcePartitions
- spark.sql.hive.metastorePartitionPruning
- spark.sql.hive.thriftServer.async
- spark.sql.hive.verifyPartitionPath
- spark.sql.hive.version
- spark.sql.inMemoryColumnarStorage.batchSize
- spark.sql.inMemoryColumnarStorage.compressed
- spark.sql.inMemoryColumnarStorage.enableVectorizedReader
- spark.sql.jsonGenerator.ignoreNullFields
- spark.sql.legacy.allowHashOnMapType
- spark.sql.mapKeyDedupPolicy
- spark.sql.maven.additionalRemoteRepositories
- spark.sql.maxPlanStringLength
- spark.sql.optimizer.dynamicPartitionPruning.enabled
- spark.sql.optimizer.excludedRules
- spark.sql.orc.columnarReaderBatchSize
- spark.sql.orc.compression.codec
- spark.sql.orc.enableVectorizedReader
- spark.sql.orc.filterPushdown
- spark.sql.orc.mergeSchema
- spark.sql.orderByOrdinal
- spark.sql.parquet.binaryAsString

- spark.sql.parquet.columnarReaderBatchSize
- spark.sql.parquet.compression.codec
- spark.sql.parquet.enableVectorizedReader
- spark.sql.parquet.filterPushdown
- spark.sql.parquet.int96AsTimestamp
- spark.sql.parquet.int96TimestampConversion
- spark.sql.parquet.mergeSchema
- spark.sql.parquet.outputTimestampType
- spark.sql.parquet.recordLevelFilter.enabled
- spark.sql.parquet.respectSummaryFiles
- spark.sql.parquet.writeLegacyFormat
- spark.sql.parser.quotedRegexColumnNames
- spark.sql.pivotMaxValues
- spark.sql.pyspark.jvmStacktrace.enabled
- spark.sql.redaction.options.regex
- spark.sql.redaction.string.regex
- spark.sql.repl.eagerEval.enabled
- spark.sql.repl.eagerEval.maxNumRows
- spark.sql.repl.eagerEval.truncate
- spark.sql.session.timeZone
- spark.sql.shuffle.partitions
- spark.sql.sources.bucketing.enabled
- spark.sql.sources.bucketing.maxBuckets
- spark.sql.sources.default
- spark.sql.sources.parallelPartitionDiscovery.threshold
- spark.sql.sources.partitionColumnTypeInference.enabled
- spark.sql.sources.partitionOverwriteMode
- spark.sql.statistics.fallBackToHdfs
- spark.sql.statistics.histogram.enabled

- spark.sql.statistics.size.autoUpdate.enabled
- spark.sql.storeAssignmentPolicy
- spark.sql.streaming.checkpointLocation
- spark.sql.streaming.continuous.epochBacklogQueueSize
- spark.sql.streaming.disabledV2Writers
- spark.sql.streaming.fileSource.cleaner.numThreads
- spark.sql.streaming.forceDeleteTempCheckpointLocation
- spark.sql.streaming.metricsEnabled
- spark.sql.streaming.multipleWatermarkPolicy
- spark.sql.streaming.noDataMicroBatches.enabled
- spark.sql.streaming.numRecentProgressUpdates
- spark.sql.streaming.stopActiveRunOnRestart
- spark.sql.streaming.stopTimeout
- spark.sql.thriftserver.scheduler.pool
- spark.sql.thriftserver.ui.retainedSessions
- spark.sql.thriftserver.ui.retainedStatements
- spark.sql.variable.substitute

Static SQL Configuration

Cấu hình SQL tĩnh là cấu hình Spark SQL xuyên phiên, bất biến. Chúng có thể được đặt với các giá trị cuối cùng bằng tệp cấu hình và các tùy chọn dòng lệnh có tiền tố --conf / -c hoặc bằng cách đặt SparkConf được sử dụng để tạo SparkSession. Người dùng bên ngoài có thể truy vấn các giá trị cấu hình sql tĩnh qua SparkSession.conf hoặc thông qua lệnh set, ví dụ: ĐẶT spark.sql.extensions ;, nhưng không thể đặt / bỏ đặt chúng.

Property Name

- spark.sql.event.truncate.length
- spark.sql.extensions
- spark.sql.hive.metastore.barrierPrefixes

- spark.sql.hive.metastore.jars
- spark.sql.hive.metastore.sharedPrefixes
- spark.sql.hive.metastore.version
- spark.sql.hive.thriftServer.singleSession
- spark.sql.legacy.sessionInitWithConfigDefaults
- spark.sql.queryExecutionListeners
- spark.sql.streaming.streamingQueryListeners
- spark.sql.streaming.ui.enabled
- spark.sql.streaming.ui.retainedProgressUpdates
- spark.sql.streaming.ui.retainedQueries
- spark.sql.ui.retainedExecutions
- spark.sql.warehouse.dir

Spark Streaming

Property Name

- spark.streaming.backpressure.enabled
- spark.streaming.backpressure.initialRate
- spark.streaming.blockInterval
- spark.streaming.receiver.maxRate
- spark.streaming.receiver.writeAheadLog.enable
- spark.streaming.unpersist
- spark.streaming.stopGracefullyOnShutdown
- spark.streaming.kafka.maxRatePerPartition
- spark.streaming.kafka.minRatePerPartition
- spark.streaming.ui.retainedBatches
- spark.streaming.driver.writeAheadLog.closeFileAfterWrite
- spark.streaming.receiver.writeAheadLog.closeFileAfterWrite

SparkR

- spark.r.numRBackendThreads
- spark.r.command
- spark.r.driver.command
- spark.r.shell.command
- spark.r.backendConnectionTimeout
- spark.r.heartBeatInterval

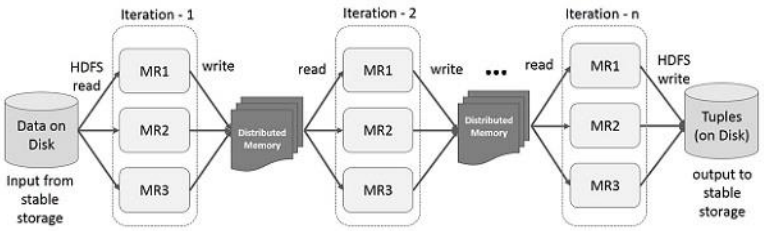
GraphX

- spark.graphx.pregel.checkpointInterval

Deploy

- spark.deploy.recoveryMode
- spark.deploy.zookeeper.url
- spark.deploy.zookeeper.dir

2. Spark RDD



Mấu chốt của Spark nằm ở RDD. Nó là một tập hợp các phần tử phân tán bất biến được phân vùng trên các nút của cụm có thể được vận hành song song với các API cấp thấp, cho phép các phép biến đổi và hành động dễ dàng.

Trường hợp sử dụng

- Trên dữ liệu phi cấu trúc, như luồng.
- Khi thao tác dữ liệu liên quan đến các cấu trúc của lập trình chức năng.
- Việc truy cập và xử lý dữ liệu không có sự áp đặt lược đồ.
- Yêu cầu chuyển đổi cấp thấp và hành động.

Các tính năng của RDD

Đa năng

Nó có thể dễ dàng và hiệu quả xử lý cả dữ liệu có cấu trúc và không cấu trúc. Nó có sẵn trong một số ngôn ngữ lập trình như Java, Scala, Python và R.

Bộ sưu tập phân tán

Nó dựa trên các hoạt động MapReduce phổ biến rộng rãi để xử lý và tạo các bộ dữ liệu lớn song song bằng cách sử dụng các thuật toán phân tán trên một cụm. Nó cho phép chúng tôi viết các tính toán song song với sự trợ giúp của các nhà khai thác cấp cao, mà không cần chi phí phân phối công việc và khả năng chịu lỗi.

Bất biến

RDD là một tập hợp các bản ghi được phân vùng. Phân vùng là một đơn vị nguyên thủy của lập trình song song trong RDD và mọi phân vùng tạo thành một phân chia dữ liệu hợp lý, không thay đổi và được tạo bằng các phép biến đổi trên các phân vùng hiện có.

Chịu lỗi

Trong trường hợp mất RDD, người ta có thể làm lại phép biến đổi trên cùng phân vùng đó và đạt được kết quả tính toán tương tự thay vì sao chép dữ liệu trên nhiều nút.

Đánh giá lười biếng

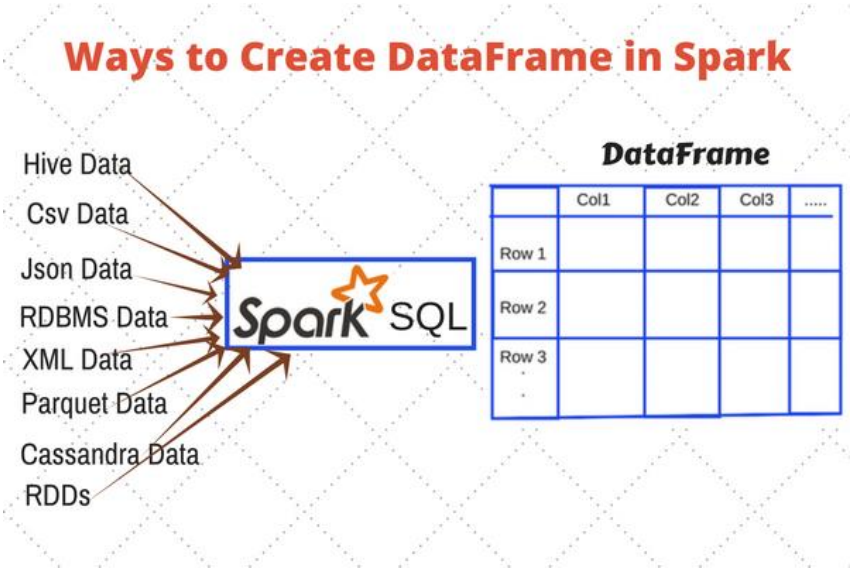
Tất cả các biến đổi là lười biếng - họ không tính toán kết quả của họ ngay lập tức. Các biến đổi được thực hiện theo yêu cầu và sau đó được trả lại cho chương trình người gọi.

Hạn chế của RDD

Không có công cụ tối ưu hóa tích hợp. Khi làm việc với dữ liệu có cấu trúc, RDD không tận dụng các trình tối ưu hóa tiên tiến của Spark (trình tối ưu hóa chất xúc tác và công cụ thực thi Vonfram). Các nhà phát triển cần tối ưu hóa từng RDD dựa trên các thuộc tính đặc tính của nó.

Ngoài ra, không giống như DataFrames và Datasets, RDD không suy ra lược đồ của dữ liệu được nhập - người dùng được yêu cầu chỉ định rõ ràng.

3. DataFrame



DataFrames là các tập hợp dữ liệu phân tán bất biến, trong đó dữ liệu được sắp xếp theo cách quan hệ - nghĩa là, các cột được đặt tên tương tự như các bảng trong cơ sở dữ liệu quan hệ. Bản chất của bộ dữ liệu là áp dụng cấu trúc trên bộ sưu tập dữ liệu phân tán để cho phép xử lý hiệu quả và dễ dàng hơn. Nó về mặt khái niệm rất tương đương với một bảng trong cơ sở dữ liệu quan hệ. Cùng với DataFrames, Spark cũng sử dụng trình tối ưu hóa chất xúc tác.

Tính năng đặc điểm

Sau đây là các tính năng nổi bật của DataFrames.

- Chúng tương đương về mặt khái niệm với một bảng trong cơ sở dữ liệu quan hệ - nhưng có tối ưu hóa phong phú hơn.

- Họ có thể xử lý các định dạng dữ liệu có cấu trúc và không cấu trúc (ví dụ Avro, CSV, ElasticSearch và Cassandra) và các hệ thống lưu trữ (ví dụ HDFS, bảng HIVE và MySQL).
- Chúng trao quyền cho các truy vấn SQL và API DataFrame.

Hạn chế của DataFrames

API DataFrame không hỗ trợ biên dịch thời gian một cách an toàn, điều này giới hạn người dùng khi thao tác dữ liệu khi không biết cấu trúc của dữ liệu.

Ngoài ra, sau khi chuyển đổi đối tượng miền thành DataFrame, người dùng không thể tạo lại nó.

TÀI LIỆU THAM KHẢO

[1]: https://spark.apache.org/docs/latest/configuration.html#cluster-managers

[2]: https://helpex.vn/article/cac-api-thong-tri-cua-spark-bo-du-lieu-dataframes-va-rdd-5c6b12a1ae03f628d053b68c