ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA HỆ THỐNG THÔNG TIN

-----📖-----

Điện Toán Đám Mây

Apache Spark-Core

Giảng viên phụ trách: Hà Lê Hoài Trung

Lớp: IS335.M12.HTCL

Sinh viên thực hiện:

Phạm Tiến Sỹ	- 18521356
Đỗ Thanh Quyền	- 18521391
Đoàn Thục Quyên	- 1852xxx

TP.HCM – 10/2021

Mục lục

1 Tổng quan

1.1 Lịch sử

Đầu tiên, vào năm 2009, Apache Spark được giới thiệu trong Phòng thí nghiệm R&D của UC Berkeley, hiện được gọi là AMPLab. Sau đó, vào năm 2010, nó trở thành mã nguồn mở theo giấy phép BSD. Hơn nữa, tia lửa đã được quyên góp cho Apache Software Foundation, vào năm 2013. Sau đó, vào năm 2014, nó trở thành dự án Apache cấp cao nhất.

1.2 Định nghĩa

- Là một nền tảng tính toán cụm đa năng với tốc độ xử lý nhanh ở phạm vi rộng.
- Được tích hợp với nhiều công cụ Dữ liệu lớn.
- Là một phần mở rộng Map reduce của Hadoop.
- Spark độc lập với Hadoop.
- Sử dụng Hadoop cho mục đích lưu trữ.
- Có nhiều ngôn ngữ lập trình: Python, Java, Scala …

1.3 Tại sao lại là Spark

- Để thực hiện xử lý hàng loạt, chúng tôi đã sử dụng **Hadoop MapReduce** .
- Ngoài ra, để thực hiện xử lý luồng, chúng tôi đã sử dụng Apache Storm / S4.
- Hơn nữa, để xử lý tương tác, chúng tôi đang sử dụng Apache Impala / Apache Tez.
- Để thực hiện xử lý đồ thị, chúng tôi đã sử dụng Neo4j / Apache Giraph.

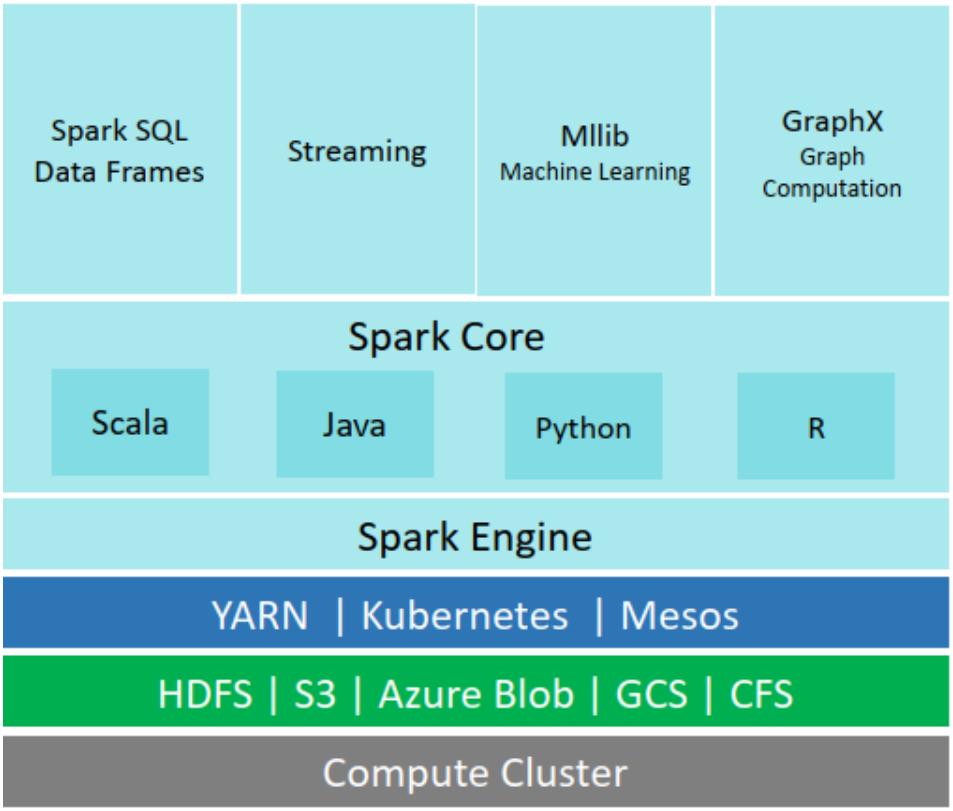
=>không có công cụ mạnh mẽ nào trong ngành có thể xử lý dữ liệu cả ở chế độ thời gian thực và chế độ hàng loạt

Khi Apache Spark ra đời đã giả quyết được các vấn đề trên.

Do đó, lập trình Apache Spark đi vào, nó là một công cụ mã nguồn mở mạnh mẽ.

Kể từ đó, nó cung cấp xử lý luồng thời gian thực, xử lý tương tác, xử lý đồ thị, xử lý trong bộ nhớ cũng như xử lý hàng loạt. Ngay cả với tốc độ rất nhanh, dễ sử dụng và giao diện chuẩn.

1.4 Hệ sinh thái của Spark



Biểu đồ này hiện thị hệ sinh thái spark kết hợp với nhau như thế nào.

Được chia thành 2 lớp:

- + Lớp dưới cùng là Spark Core => đây là lớp giao diện lập trình, cung cấp 4 ngôn ngữ chính
- + Lớp trên là tập hớp các thư viện, API Bản thân Spark có hai phần
- + Một công cụ máy tính phân tán
- + Một tập hợp các API core

Và toàn bộ 2 phần trong spark core chạy trên một cụm máy tính để cung cấp cho bạn khả năng xử lý dữ liệu phân tán.

chỉ cung cấp cho bạn một khung xử lý dữ liệu. bạn sẽ cần một trình quản lý cụm.

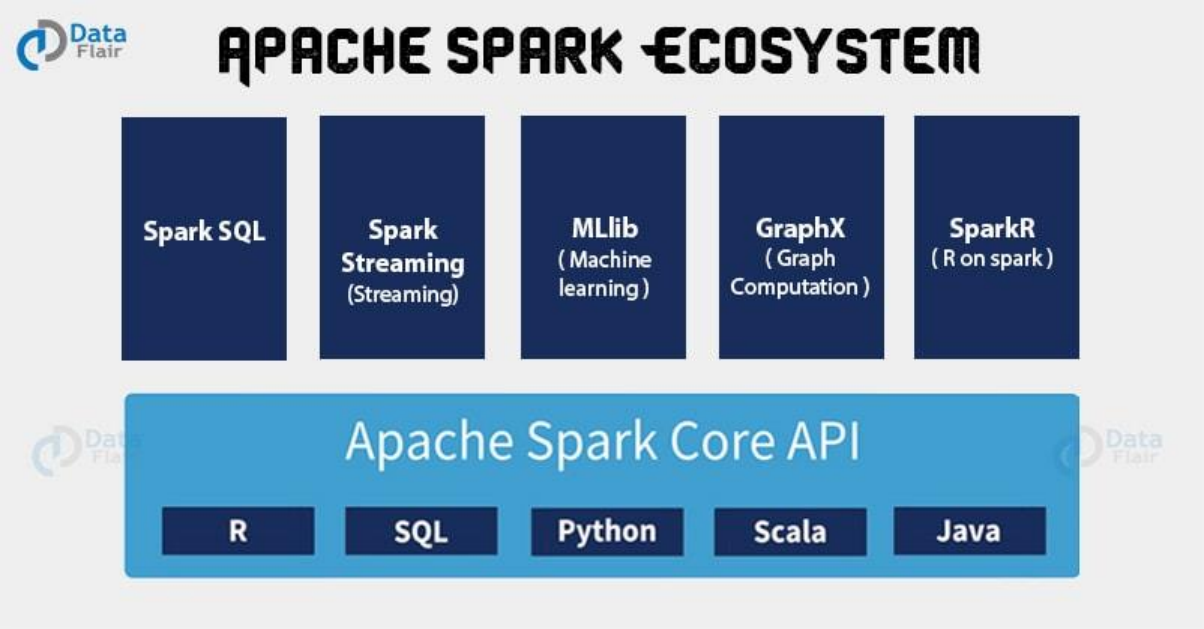
Tuy nhiên spark không có quản lý cụm => cần trính quản lý cụm Spark

ban đầu dựa trên Hadoop MR và nó đã được chấp nhận trong nền tảng Hadoop. trình quản lý tài nguyên Hadoop YARN là trình quản lý cụm được sử dụng phổ biến nhất cho Spark.

Các tuy chọn khác như Mesos Kubernetes, Standalone Cluster Manager

Spark cũng không đi kèm với hệ thống lưu trữ tích hợp. Và nó cho phép bạn xử lý dữ liệu được lưu trữ trong nhiều hệ thống lưu trữ. Một số hệ thống lưu trữ phổ biến nhất là HDFS, Amazon S3, Azure Blob, Google Cloud Storagevà hệ thống tệp Cassandra.

1.5 Các thành phần trên Spark



1.5.1 Spark Core

Tất cả các chức năng được cung cấp bởi Apache Spark đều được xây dựng trên phần đầu của Spark Core . Nó mang lại tốc độ bằng cách cung cấp khả năng tính toán trong bộ nhớ . Do đó Spark Core là nền tảng của quá trình xử lý song song và phân tán của tập dữ liệu khổng lồ.

Các tính năng chính của Apache Spark Core là:

- Nó phụ trách các chức năng I / O thiết yếu.
- Có ý nghĩa trong việc lập trình và quan sát vai trò của **cụm Spark** .
- Điều động công việc.
- Phục hồi lỗi.
- Nó vượt qua khó khăn của **MapReduce** bằng cách sử dụng tính toán trong bộ nhớ.

1.5.2 Spark SQL

Là một trong năm thành phần chính của Spark được phát triển cho việc sử lý dữ liệu có cấu trúc (structured data processing). Chúng ta có thể tương tác với SparkSQL thông qua SQL, DataFrames API hoặc Datasets API.

DataFrame là một API bậc cao hơn RDD được Spark giới thiệu vào năm 2013 (từ Apache Spark 1.3). Tương tự như RDD, dữ liệu trong DataFrame cũng được quản lý theo kiểu phân tán và không thể thay đổi (immutable distributed). Tuy nhiên dữ liệu này được sắp sếp theo các cột, tương tự như trong Relation Database. DataFrame được phát triển để giúp người dùng có thể dễ dàng thực hiện các thao tác xử lý dữ liệu cũng như làm tăng đáng kể hiệu quả sử lý của hệ thống.

1.5.3 Spark Streaming

Nó là một tiện ích bổ sung cho API Spark cốt lõi cho phép xử lý luồng dữ liệu trực tiếp có thể mở rộng, thông lượng cao, chịu được lỗi. Spark có thể truy cập dữ liệu từ

các nguồn như **Kafka** , **Flume , Kinesis** hoặc **TCP socket.** Nó có thể hoạt động bằng cách sử dụng các thuật toán khác nhau. Cuối cùng, dữ liệu nhận được sẽ được cung cấp cho hệ thống tệp, cơ sở dữ liệu và trang tổng quan trực tiếp. Spark sử dụng *Micro-batching* để phát trực tuyến trong thời gian thực.
Vi phân lô là một kỹ thuật cho phép một quá trình hoặc nhiệm vụ xử lý một luồng như một chuỗi các lô dữ liệu nhỏ. Do đó Spark Streaming, nhóm dữ liệu trực tiếp thành các lô nhỏ. Sau đó, nó chuyển nó đến hệ thống lô để xử lý. Nó cũng cung cấp các đặc tính chịu lỗi.

1.5.4 Spark Mlib

MLlib trong Spark là một thư viện Máy học có thể mở rộng thảo luận về cả thuật toán chất lượng cao và tốc độ cao.
Động cơ đằng sau việc tạo MLlib là làm cho việc học máy có thể mở rộng và dễ dàng. Nó chứa các thư viện học máy có triển khai các thuật toán học máy khác nhau. Ví dụ, *phân cụm, hồi quy, phân loại và lọc cộng tác*. Một số nguyên thủy máy học cấp thấp hơn như thuật toán tối ưu hóa độ dốc gốc chung cũng có trong MLlib. Trong phiên bản Spark 2.0, API dựa trên RDD trong gói *spark.mllib được* nhập vào chế độ bảo trì. Trong bản phát hành này, API dựa trên DataFrame là API máy học chính cho Spark. Vì vậy, từ bây giờ MLlib sẽ không thêm bất kỳ tính năng mới nào vào API dựa trên RDD.

1.5.5 Spark GraphX

GraphX trong Spark là API cho đồ thị và thực hiện song song đồ thị. Nó là công cụ phân tích đồ thị mạng và lưu trữ dữ liệu. *Cũng có thể phân cụm, phân loại, duyệt, tìm kiếm và tìm đường dẫn* trong biểu đồ. Hơn nữa, GraphX mở rộng Spark RDD bằng cách đưa vào ánh sáng một trừu tượng Đồ thị mới: một đồ thị có hướng với các thuộc tính gắn với mỗi đỉnh và cạnh.
GraphX cũng tối ưu hóa cách mà chúng ta có thể biểu diễn đỉnh và cạnh khi chúng là kiểu dữ liệu nguyên thủy. Để hỗ trợ tính toán đồ thị, nó hỗ trợ các toán tử cơ bản (ví dụ: đồ thị con, nối các Vertices và tổng hợp Thông báo) cũng như một biến thể được tối ưu hóa của *Pregel API* .

1.5.6 SparkR

SparkR là phiên bản Apache Spark 1.4. Thành phần quan trọng của SparkR là SparkR DataFrame. DataFrames là một cấu trúc dữ liệu cơ bản để xử lý dữ liệu trong **R.** Khái niệm DataFrames mở rộng sang các ngôn ngữ khác với các thư viện như *Pandas,* v.v. R cũng cung cấp các phương tiện phần mềm để thao tác dữ liệu, tính toán và hiển thị đồ họa. Do đó, ý tưởng chính đằng sau SparkR là khám phá các kỹ thuật khác nhau để tích hợp khả năng sử dụng của R với khả năng mở rộng của Spark. Đó là gói R cung cấp giao diện người dùng nhẹ để sử dụng Apache Spark từ R.

1.6 Ưu điểm

- **Advanced Analytics:** Spark không chỉ hỗ trợ “Map” và “Reduce”, nó còn hỗ trợ truy vấn SQL, Streaming data, Machine learning và các thuật toán đồ thị.

- **Speed**: gấp 100 trêm ứng dụng và 10 lần trên đĩa.
- **Multiple languages:** Cung cấp built-in API phổ biến như Java, Scala, Python, R. Bên cạnh đó Spark còn cung cấp nhiều high-level operators cho việc truy vấn dữ liệu…

1.7 Nhược điểm

- Spark không có hệ thống Filesytem riêng.
- Đòi hỏi rất nhiều RAM để chạy trong bộ nhớ.
- Spark Streaming không thực sự real-time.
- Việc tối ưu hóa, chỉnh sửa dữ liệu cần phải có kinh nghiệm và cần thực hiện thủ công.

1.8 Ứng dụng

- Ngành tài chính.
- Thương mại điện tử.
- Truyền thông và giải trí.
- Du lịch.
- Chăm sóc sức khỏe.

2 Spark Core & RDD

2.1 Distributed computing

Giả sử các máy tính màu xanh chứa dữ liệu và phân tán trong 1 cụm, thông thường, khi muốn xử lý dữ liệu này, ta sẽ "kéo" dữ liệu liên quan từ các máy màu xanh về máy mày đen để xử lý. Tuy nhiên, với Spark sẽ ngược lại, ta sẽ gửi code xử lý từ máy màu

đen đến các máy màu xanh, sau đó dữ liệu sẽ được xử lý trên các máy màu xanh, rồi lại lưu luôn tại máy màu xanh. Vừa không mất công chuyển đổi dữ liệu qua lại, lại giảm nhẹ gánh nặng tính toán lên 1 máy tính.

2.2 RDD

- RDD là một **đơn vị dữ liệu** cơ bản của Apache Spark mà bị chia thành các partitions và thực thi trên các nodes khác nhau của cluster.
- RDD là một **đơn vị dữ liệu** cơ bản của Apache Spark mà bị chia thành các partitions và thực thi trên các nodes khác nhau của cluster.
- Một RDD bao gồm nhiều partition nhỏ, mỗi partition này đại diện cho 1 phần dữ liệu phân tán. Khái niệm partition là logical, tức là 1 node xử lý có thể chứa nhiều hơn 1 RDD partition. Theo mặc định, dữ liệu các partitions sẽ lưu trên memory.

2.3 Lazy Evaluation

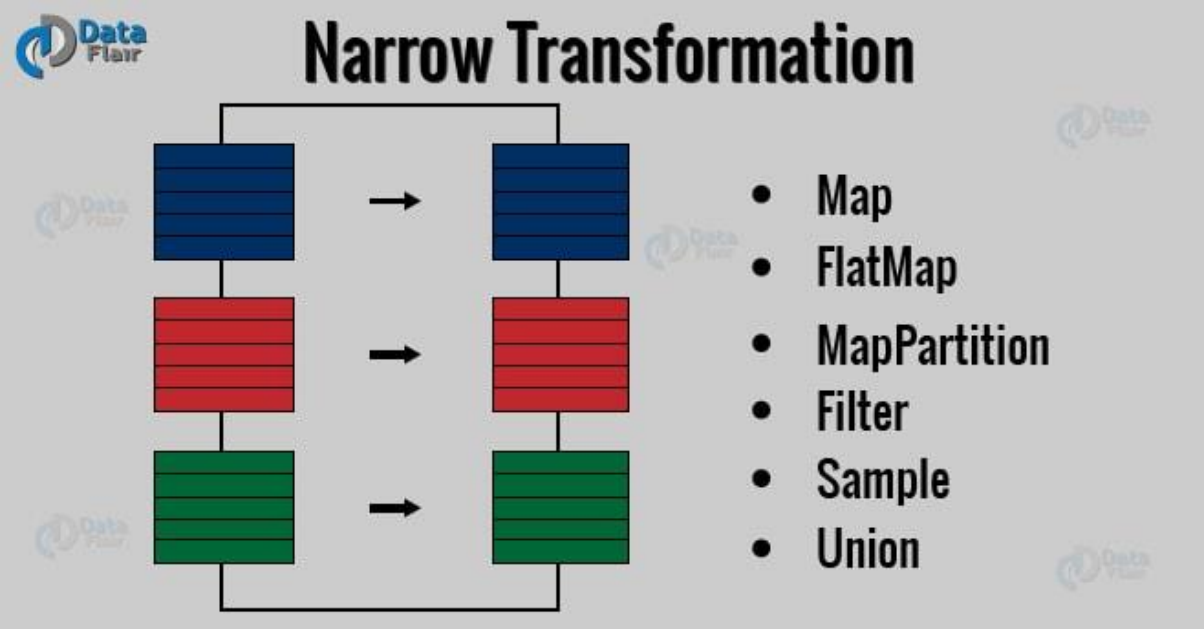
Khi thực thi, việc gọi các transformations, Spark sẽ không ngay lập tức thực thi các tính toán mà sẽ lưu lại thành 1 lineage, tức là tập hợp các biến đổi từ RDD này thành RDD khác qua mỗi transformation. Khi có 1 action được gọi, Spark lúc này mới thực sự thực hiện các biến đổi để trả ra kết quả.

2.4 Transformation

Spark Transformation là một chức năng tạo ra RDD mới từ các RDD hiện có. Nó nhận RDD làm đầu vào và tạo ra một hoặc nhiều RDD làm đầu ra. Mỗi lần nó tạo ra RDD mới khi chúng ta áp dụng bất kỳ chuyển đổi nào. Do đó, các RDD đầu vào như vậy, không thể thay đổi vì RDD về bản chất là bất biến.

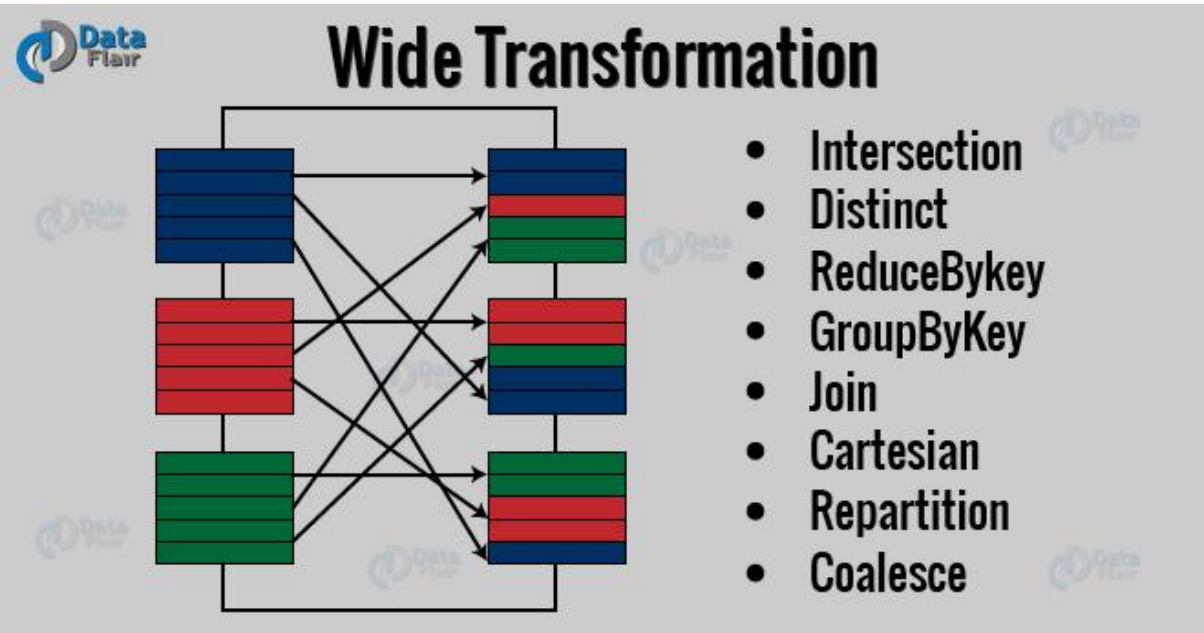
2.4.1 Narrow Transformation

Tất cả các phần tử được yêu cầu để tính toán các bản ghi trong phân vùng đơn lẻ sống trong phân vùng duy nhất của RDD mẹ. Một tập hợp con giới hạn của phân vùng được sử dụng để tính toán kết quả. Các phép biến đổi hẹp *là kết quả của* map (), filter ().



2.4.2 Wide Transformation

Tất cả các phần tử được yêu cầu để tính toán các bản ghi trong phân vùng duy nhất có thể nằm trong nhiều phân vùng của RDD mẹ. Phân vùng có thể nằm trong nhiều phân vùng của RDD mẹ. *Các phép biến đổi rộng* là kết quả của *groupbyKey () và ReducebyKey ()*.



2.5 Action

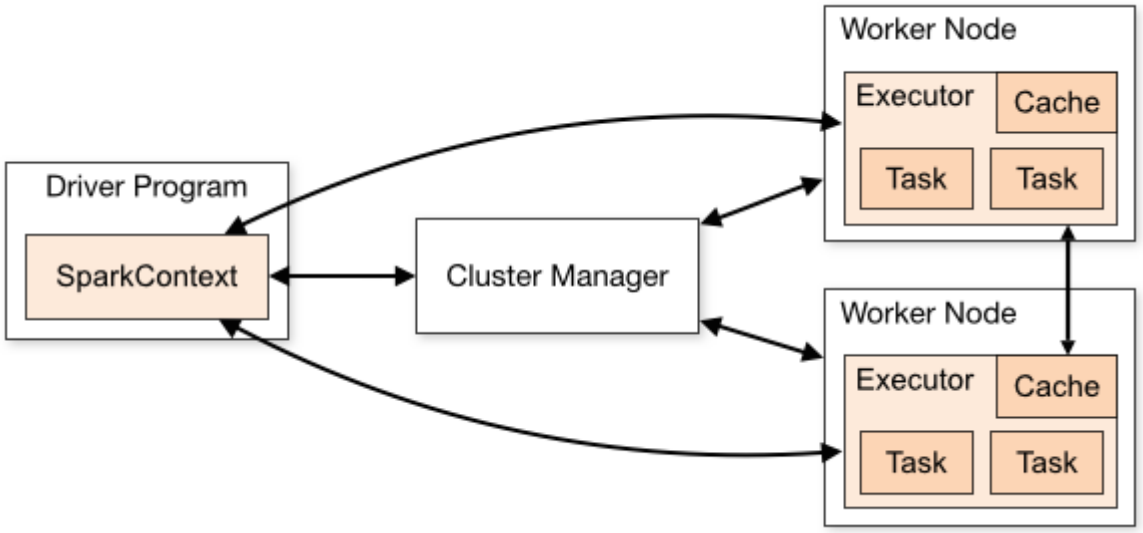
Là một trong những cách gửi dữ liệu từ *Executer* đến *trình điều khiển*. Người thừa hành là những tác nhân có trách nhiệm thực thi một nhiệm vụ. Trong khi trình điều khiển là một quy trình JVM điều phối các công nhân và thực thi nhiệm vụ.

3 Cluster

3.1 Giới thiệu

Các ứng dụng Spark chạy như một tập hợp các quy trình độc lập trên một cụm, được điều phối bởi đối tượng SparkContext trong chương trình chính (được gọi là chương trình điều khiển).

Cụ thể, để chạy trên một cụm, SparkContext có thể kết nối với một số loại trình quản lý cụm (có thể là trình quản lý cụm độc lập của Spark, Mesos hoặc YARN), phân bổ tài nguyên trên các ứng dụng. Sau khi được kết nối, Spark có được các trình thực thi trên các nút trong cụm, là các quy trình chạy tính toán và lưu trữ dữ liệu cho ứng dụng của bạn. Tiếp theo, nó sẽ gửi mã ứng dụng của bạn (được xác định bởi các tệp JAR hoặc Python được chuyển đến SparkContext) cho những người thực thi. Cuối cùng, SparkContext gửi các tác vụ đến những người thực thi để chạy.



(hình ảnh minh hoạ cách hoạt động của cluster)

một số điều hữu ích cần lưu ý về kiến trúc này:

- Mỗi ứng dụng có các quy trình thực thi của riêng nó, các quy trình này duy trì trong suốt thời gian của toàn bộ ứng dụng và chạy các tác vụ trong nhiều luồng. Điều này có lợi ích là cô lập các ứng dụng với nhau, ở cả phía lập lịch (mỗi trình điều khiển lập lịch cho các tác vụ riêng của mình) và phía thực thi (các tác vụ từ các ứng dụng khác nhau chạy trong các JVM khác nhau). Tuy nhiên, điều đó cũng có nghĩa là dữ liệu không thể được chia sẻ trên các ứng dụng Spark khác nhau (các phiên bản của SparkContext) mà không ghi dữ liệu đó vào hệ thống lưu trữ bên ngoài.
- Spark là bất khả tri đối với người quản lý cụm cơ bản. Miễn là nó có thể có được các quy trình thực thi và các quy trình này giao tiếp với nhau, thì việc chạy nó tương đối dễ dàng ngay cả trên trình quản lý cụm cũng hỗ trợ các ứng dụng khác (ví dụ: Mesos / YARN).

- Chương trình trình điều khiển phải lắng nghe và chấp nhận các kết nối đến từ các trình thực thi của nó trong suốt thời gian tồn tại của nó (ví dụ: xem spark.driver.port trong phần cấu hình mạng). Như vậy, chương trình trình điều khiển phải có địa chỉ mạng từ các nút công nhân.
- Vì trình điều khiển lập lịch các tác vụ trên cụm, nó phải được chạy gần các nút công nhân, tốt nhất là trên cùng một mạng cục bộ. Nếu bạn muốn gửi yêu cầu đến cụm từ xa, tốt hơn nên mở RPC tới trình điều khiển và yêu cầu nó gửi các hoạt động từ gần đó hơn là chạy trình điều khiển ở xa các nút công nhân.

3.2 Các loại quản lí cụm

Trong hệ thống Apache Spark hiện tại hỗ trợ một số quy trình quản lý cụm:

- **Standalone**: là trình quản lí cụm đơn giản đi kèm với Spark giúp dễ dàng thiết lập.
- **Apache Mesos**: là một trình quản lí cụm chung có thể chạy Hadoop MapReduce và các ứng dụng dịch vụ.
- **Hadoop YARN**: là trình quản lí tài nguyên trong Hadoop 2.

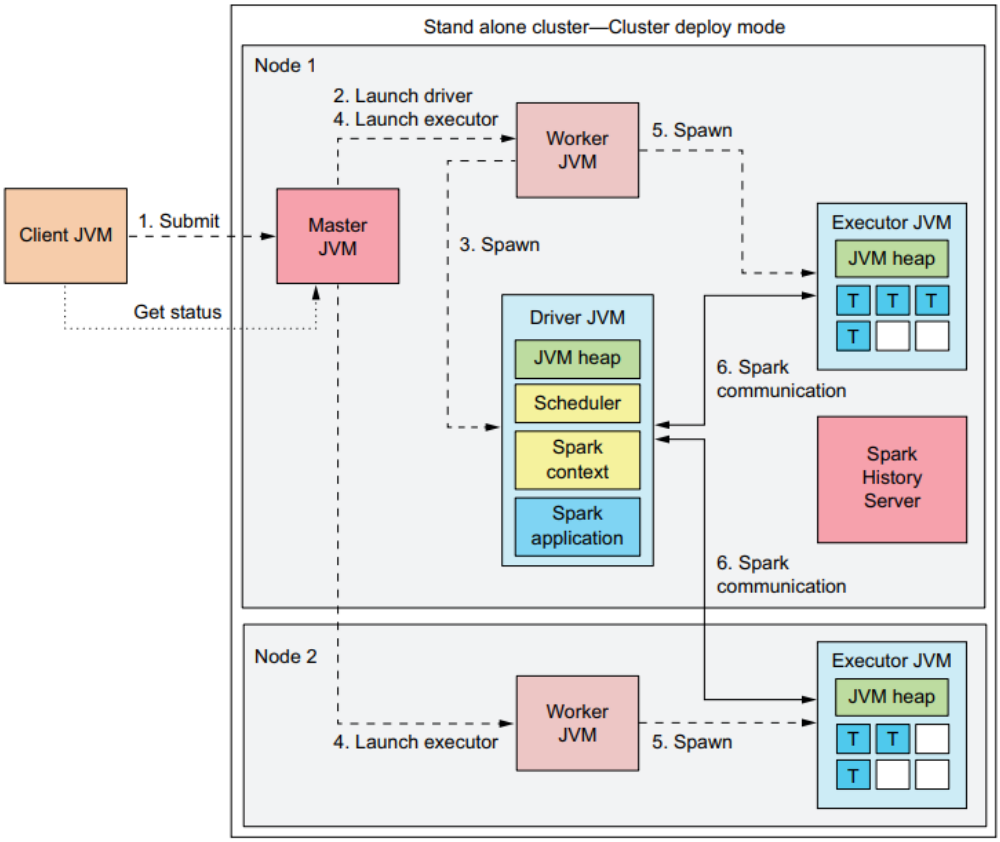
3.2.1 Standalone

Standalone cluster của spark có một kiến trúc đơn giản dễ cài đặt và cấu hình, bởi vì nó được xây dựng và tối ưu hoá đặc biệt cho Spark.

Vì thế nên nó không được hỗ trợ giao tiếp với HDFS được bảo mật bằng giao thức xác thực Kerberos.

Standalone bao gồm Master và Worker (hay còn gọi là Slave). Nó cho phép các ứng dụng được chạy và lặp lịch cho các tài nguyên của worker (với các nhân CPU có sẵn) và nó phải được cài đặt trên tất cả các nút trong cụm.

Quá trình chạy của Standalone trong Cluster deploy mode:



Bước 1: người dùng sẽ submit application lên master để thực thi.

Bước 2: Master sẽ hướng dẫn 1 worker có sẵn khởi động trình điều khiển Driver.

Bước 3: Worker sẽ tạo ra một Driver JVM

Bước 4: Master sẽ hướng dẫn cả 2 worker khởi chạy thực thi cho application.

Bước 5: Worker sẽ sinh ra các executor JVM

Bước 6: Driver và executor sẽ giao tiếp độc lập với các quy trình quản lý cụm để thực thi application.

Khởi chạy cluster trên shell:

- start-master.sh bắt đầu một tiến trình master.
- start-slaves.sh bắt đầu các worker được cấu hình.
- start-all.sh chạy cả master và worker.

Nếu muốn kết thúc chỉ cần thay start bằng stop ở các lệnh trên.

- stop-master.sh để kết thúc một tiến trình master.
- stop-slaves.sh kết thúc các worker được cấu hình.
- stop-all.sh kết thúc cả master và worker.

3.2.2 Apache Mesos

Mesos cung cấp nhân hệ thống phân tán và cung cấp tài nguyên cụm hàng hóa cho các ứng dụng, giống như nhân Linux quản lý tài nguyên của một máy tính và cung cấp chúng cho các ứng dụng chạy trên một máy. Mesos hỗ trợ các ứng dụng được viết bằng Java, C, C ++ và Python. Với phiên bản 0.20, Mesos có thể chạy các vùng chứa Docker, là các gói chứa tất cả các thư viện và cấu hình mà ứng dụng cần để chạy. Với sự hỗ trợ của Docker, bạn có thể chạy

Mesos trên hầu như bất kỳ ứng dụng nào có thể chạy trong vùng chứa Docker. Kể từ Spark 1.4, Spark cũng có thể chạy trên Mesos trong Docker container.

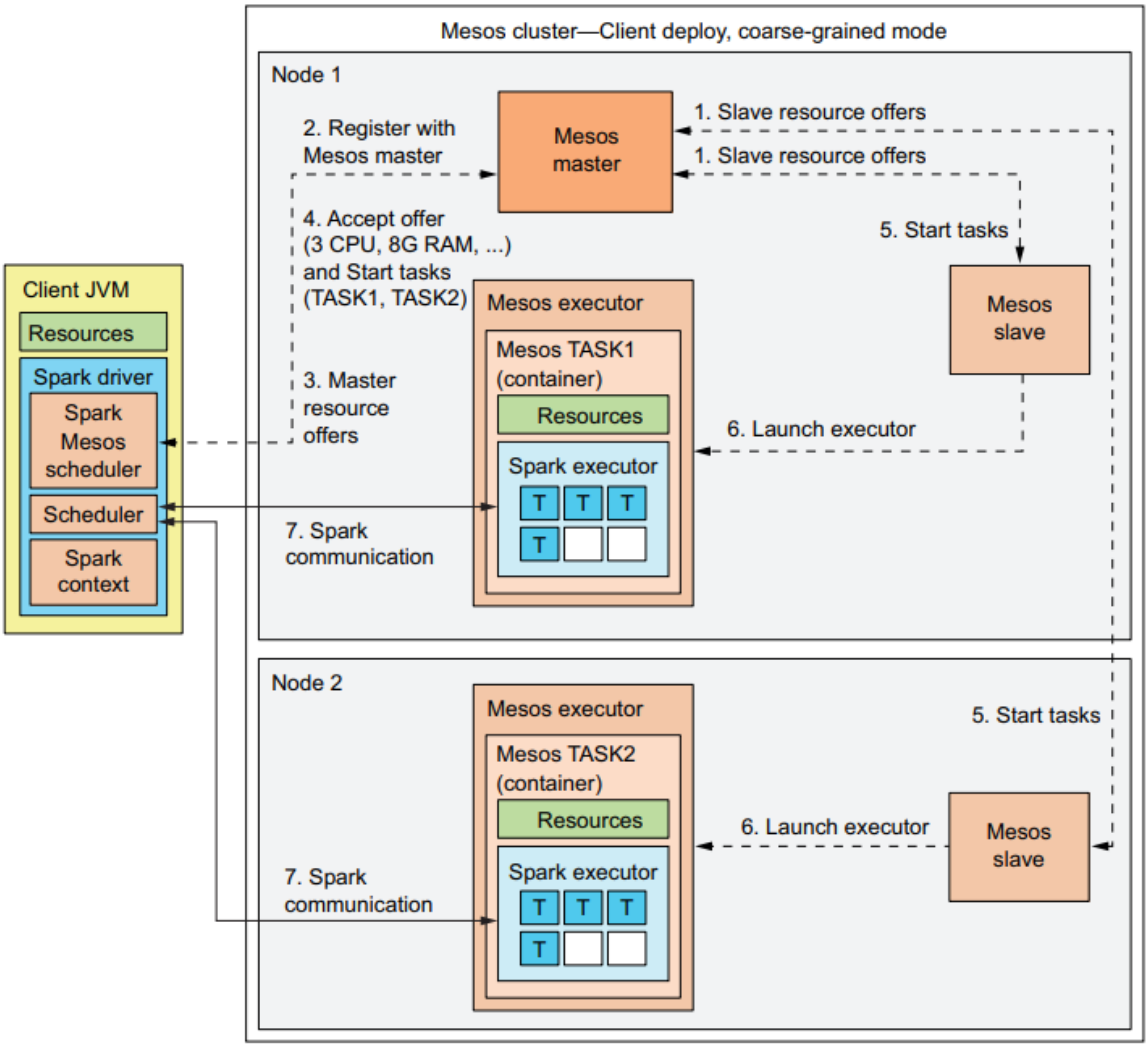
Một số điểm mà Mesos có thể sử dụng một số cải tiến là bảo mật và hỗ trợ để chạy các ứng dụng trạng thái (những ứng dụng sử dụng lưu trữ liên tục, chẳng hạn như cơ sở dữ liệu). Với phiên bản hiện tại (1.0.1), không nên chạy các ứng dụng trạng thái trên Mesos. Cộng đồng cũng đang nỗ lực hỗ trợ các trường hợp sử dụng này. Ngoài ra, xác thực dựa trên Kerberos vẫn chưa được hỗ trợ https://issues.apache.org/jira/browse/MESOS-907). Tuy nhiên, các ứng dụng có thể cung cấp xác thực bằng cách sử dụng Lớp bảo mật và Xác thực đơn giản (SASL, một khuôn khổ xác thực và bảo mật dữ liệu được sử dụng rộng rãi) và giao tiếp nội bộ được bảo mật bằng Lớp cổng bảo mật (SSL). Phân bổ động (được giải thích trong phần 12.1.10), trước đây chỉ dành riêng cho YARN, khả dụng trên Mesos với Spark 1.5. Để bắt đầu khám phá việc chạy Spark trên Mesos, trước tiên chúng ta sẽ xem xét kỹ hơn kiến trúc Mesos. Sau đó, chúng ta sẽ thấy cách chạy Spark trên Mesos khác với việc chạy nó trên YARN và trong các cụm Spark độc lập.

Kiến trúc của Apache Mesos:

So sánh kiến trúc Mesos với một cụm độc lập Spark thì đơn giản hơn so với YARN. Các thành phần cơ bản của Mesos — master, slave và ứng dụng (hoặc framework, theo thuật ngữ Mesos). Như trường hợp với một cụm độc lập Spark, một tổng thể Mesos lập lịch tài nguyên nô lệ giữa các ứng dụng muốn sử dụng họ. Các nô lệ khởi chạy các trình thực thi của ứng dụng để thực thi các tác vụ. Càng xa càng tốt. Mesos mạnh hơn một cụm Spark độc lập và khác với nó ở một số điểm quan trọng. Đầu tiên, nó có thể lên lịch cho các loại ứng dụng khác ngoài Spark (các ứng dụng Java, Scala, C, C ++ và Python). Nó cũng là có khả năng lên lịch không gian đĩa, cổng mạng và thậm chí cả tài nguyên tùy chỉnh (không chỉ CPU và bộ nhớ). Và thay vì các ứng dụng yêu cầu tài nguyên từ cụm (từ tổng thể của nó), một cụm Mesos cung cấp tài nguyên cho các ứng dụng, mà chúng có thể chấp nhận hoặc từ chối.

Các khung chạy trên Mesos (chẳng hạn như các ứng dụng Spark) bao gồm hai thành phần: một bộ lập lịch và một bộ thực thi. Bộ lập lịch trình chấp nhận hoặc từ chối các tài nguyên được cung cấp bởi Mesos master và tự động khởi động các trình thực thi Mesos trên các nô lệ. Người thực thi Mesos chạy các tác vụ theo yêu cầu của bộ lập lịch của khuôn khổ.

Quá trình chạy của Apache Mesos trong Client deloy mode & coarse-grained modes:



Bước 1: Các nô lệ Mesos cung cấp tài nguyên của họ cho chủ nhân.

Bước 2: Bộ lập lịch biểu dành riêng cho Mesos của Spark, chạy trong trình điều khiển (ví dụ: lệnh gửi Spark) đăng ký với Mesos master.

Bước 3: Đến lượt nó, Mesos master cung cấp các tài nguyên có sẵn cho bộ lập lịch Spark Mesos (điều này xảy ra liên tục: theo mặc định, mỗi giây trong khi framework còn hoạt động).

Bước 4: Trình lập lịch Mesos của Spark chấp nhận một số tài nguyên và gửi danh sách tài nguyên, cùng với danh sách các tác vụ mà nó muốn thực hiện bằng cách sử dụng các tài nguyên đó tới Mesos master.

Bước 5: Chủ yêu cầu các nô lệ bắt đầu các nhiệm vụ với các tài nguyên được yêu cầu.

Bước 6: Các nô lệ khởi chạy các trình thực thi (trong trường hợp này là Mesos's Command Executor), khởi chạy các trình thực thi Spark (sử dụng lệnh được cung cấp) trong các vùng chứa tác vụ.

Bước 7: Người thực thi Spark kết nối với trình điều khiển Spark và tự do giao tiếp với nó, thực hiện các tác vụ Spark như bình thường.

3.2.3 Hadoop YARN

YARN (là viết tắt của một trình thương lượng tài nguyên khác) là thế hệ mới của công cụ thực thi MapReduce của Hadoop. Không giống như công cụ MapReduce trước đây, chỉ có thể chạy các công việc MapReduce, YARN có thể chạy các loại chương trình khác (chẳng hạn như Spark).

Hầu hết các bản cài đặt Hadoop đều đã được định cấu hình YARN cùng với HDFS, vì vậy YARN là công cụ thực thi tự nhiên nhất cho nhiều người dùng Spark tiềm năng và hiện tại. Spark được thiết kế để không phụ thuộc vào trình quản lý cụm cơ bản và đang chạy

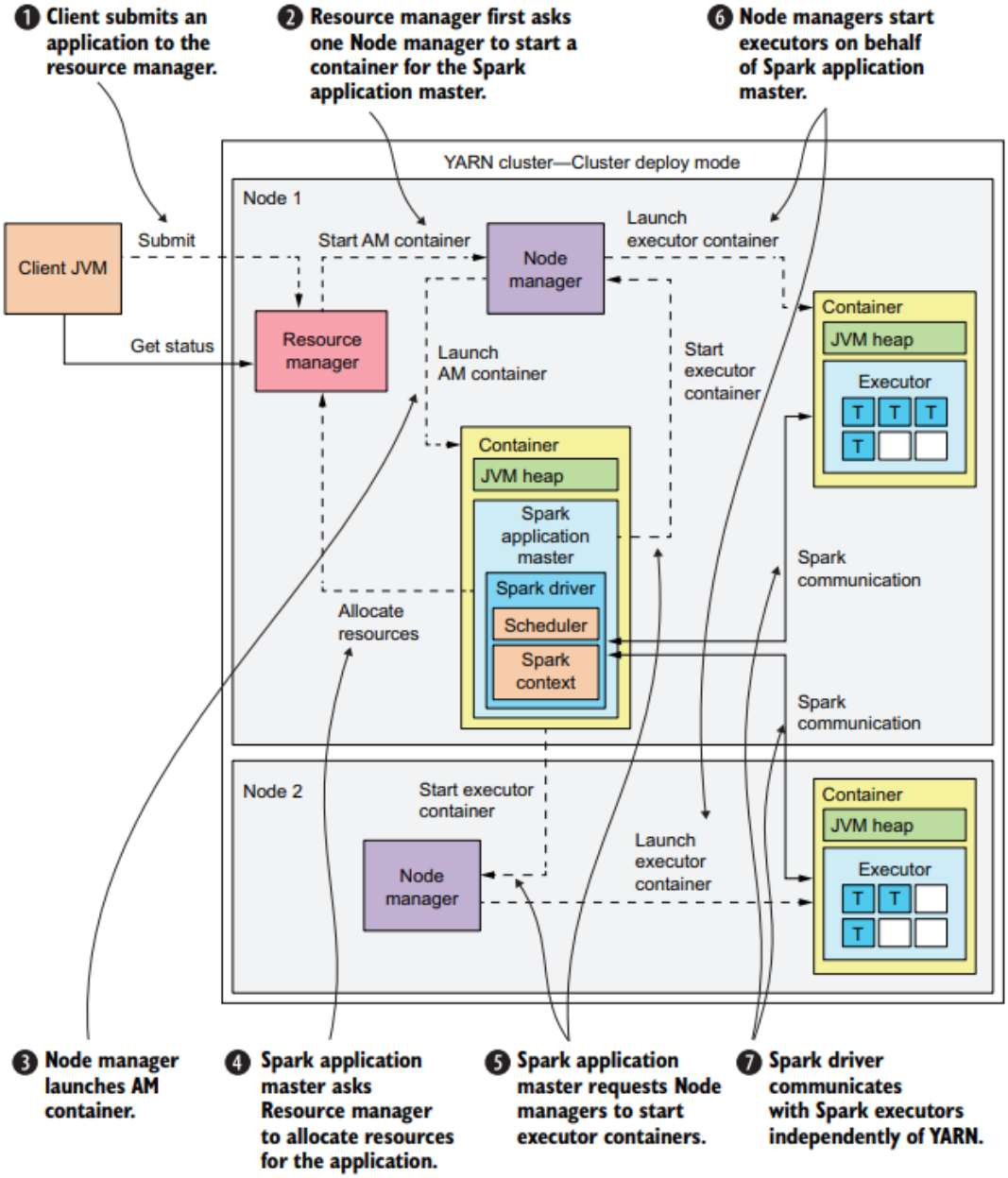
Các ứng dụng Spark trên YARN không khác nhiều so với việc chạy chúng trên các trình quản lý cụm khác, nhưng có một số điểm khác biệt mà bạn cần lưu ý. Chúng ta sẽ xem xét những điểm khác biệt đó ở đây.

Chúng ta sẽ bắt đầu khám phá việc chạy Spark trên YARN bằng cách xem xét kiến trúc YARN trước. Sau đó, chúng tôi sẽ mô tả cách gửi ứng dụng Spark tới YARN, sau đó giải thích sự khác biệt giữa việc chạy ứng dụng Spark trên YARN so với một cụm Spark độc lập.

Kiến trúc của Hadoop YARN

Kiến trúc YARN cơ bản tương tự như kiến trúc cụm độc lập của Spark. Nó là các thành phần chính là trình quản lý tài nguyên (nó có thể được ví như quy trình chính của Spark) cho mỗi cụm và trình quản lý nút (tương tự như quy trình công nhân của Spark) cho mỗi nút trong cụm. Không giống như chạy trên cụm độc lập của Spark, các ứng dụng trên YARN chạy trong vùng chứa (JVM xử lý tài nguyên CPU và bộ nhớ được cấp). Một ứng dụng tổng thể cho mỗi ứng dụng là một thành phần đặc biệt. Đang chạy trong vùng chứa của chính nó, nó có trách nhiệm yêu cầu tài nguyên ứng dụng từ người quản lý tài nguyên. Khi Spark đang chạy trên YARN, quá trình trình điều khiển Spark hoạt động như ứng dụng YARN chính. Người quản lý nút theo dõi tài nguyên được sử dụng bởi vùng chứa và báo cáo cho người quản lý tài nguyên.

Quá trình chạy YARN cluster trong Cluster deloy mode:


1 Client submits an application to the resource manager.
2 Resource manager first asks one Node manager to start a container for the Spark application master.
6 Node managers start executors on behalf of Spark application master.
YARN cluster—Cluster deploy mode
Node 1
Client JVM
Submit
Start AM container
Node manager
Launch executor container
Container
JVM heap
Executor
T T T
T
Get status
Resource manager
Launch AM container
Start executor container
Container
JVM heap
Spark application master
Spark driver
Scheduler
Spark context
Spark communication
Allocate resources
Spark communication
Node 2
Start executor container
Node manager
Launch executor container
Container
JVM heap
Executor
T T
T
3 Node manager launches AM container.
4 Spark application master asks Resource manager to allocate resources for the application.
5 Spark application master requests Node managers to start executor containers.
7 Spark driver communicates with Spark executors independently of YARN.