
Lời nói đầu

Trong một thời gian rất ngắn, Apache Spark đã nổi lên như một công cụ xử lý dữ liệu lớn thế hệ tiếp theo và đang được áp dụng trong toàn ngành nhanh hơn bao giờ hết. Spark cải thiện Hadoop MapReduce, công cụ đã giúp châm ngòi cho cuộc cách mạng dữ liệu lớn, ở một số khía cạnh chính: nhanh hơn nhiều, dễ sử dụng hơn nhiều do có API phong phú và vượt xa các ứng dụng hàng loạt để hỗ trợ nhiều khối lượng công việc khác nhau, bao gồm truy vấn tương tác, phát trực tuyến, học máy và xử lý đồ thị.

Tôi rất vinh dự được tham gia chặt chẽ vào quá trình phát triển Spark từ bản vẽ cho đến dự án nguồn mở dữ liệu lớn năng động nhất hiện nay và là một trong những dự án Apache năng động nhất! Vì vậy, tôi đặc biệt vui mừng khi thấy Matei Zaharia, người sáng tạo ra Spark, hợp tác với các nhà phát triển Spark lâu năm khác là Patrick Wendell, Andy Konwinski và Holden Karau để viết cuốn sách này.

Với sự gia tăng nhanh chóng về mức độ phổ biến của Spark, một mối quan tâm lớn là thiếu tài liệu tham khảo tốt. Cuốn sách này đi một chặng đường dài để giải quyết mối quan tâm này, với 11 chương và hàng chục ví dụ chi tiết được thiết kế cho các nhà khoa học dữ liệu, sinh viên và nhà phát triển muốn tìm hiểu về Spark. Nó được viết để dễ tiếp cận đối với những độc giả không có kiến thức nền tảng về dữ liệu lớn, khiến nó trở thành một nơi tuyệt vời để bắt đầu tìm hiểu về lĩnh vực này nói chung. Tôi hy vọng rằng nhiều năm sau, bạn và những độc giả khác sẽ nhớ đến cuốn sách này như một *cái* cuốn sách giới thiệu cho bạn lĩnh vực mới thú vị này.

— Ion Stoica, Tổng giám đốc điều hành của Databricks và Đồng giám đốc, AMPLab, UC Berkeley

Lời nói đầu

Khi phân tích dữ liệu song song trở nên phổ biến, các học viên trong nhiều lĩnh vực đã tìm kiếm các công cụ dễ dàng hơn cho nhiệm vụ này. Apache Spark đã nhanh chóng nổi lên như một trong những MapReduce mở rộng và tổng quát phổ biến nhất. Spark cung cấp ba lợi ích chính. Đầu tiên, nó dễ sử dụng—bạn có thể phát triển các ứng dụng trên máy tính xách tay của mình, sử dụng API cấp cao cho phép bạn tập trung vào nội dung tính toán của mình. Thứ hai, Spark nhanh, cho phép sử dụng tương tác và các thuật toán phức tạp. Và thứ ba, Spark là một *tổng quan* engine, cho phép bạn kết hợp nhiều loại tính toán (ví dụ: truy vấn SQL, xử lý văn bản và học máy) mà trước đây có thể yêu cầu các engine khác nhau. Các tính năng này khiến Spark trở thành điểm khởi đầu tuyệt vời để tìm hiểu về Dữ liệu lớn nói chung.

Cuốn sách giới thiệu này nhằm giúp bạn bắt đầu và chạy Spark một cách nhanh chóng. Bạn sẽ học cách tải xuống và chạy Spark trên máy tính xách tay của mình và sử dụng nó một cách tương tác để tìm hiểu API. Khi đã ở đó, chúng ta sẽ đề cập đến các chi tiết về các hoạt động khả dụng và thực thi phân tán. Cuối cùng, bạn sẽ được tham quan các thư viện cấp cao hơn được tích hợp vào Spark, bao gồm các thư viện cho máy học, xử lý luồng và SQL. Chúng tôi hy vọng rằng cuốn sách này cung cấp cho bạn các công cụ để nhanh chóng giải quyết các vấn đề phân tích dữ liệu, cho dù bạn làm như vậy trên một máy hay hàng trăm máy.

Khán giả

Cuốn sách này hướng đến các nhà khoa học dữ liệu và kỹ sư. Chúng tôi chọn hai nhóm này vì họ có thể hưởng lợi nhiều nhất khi sử dụng Spark để mở rộng phạm vi các vấn đề mà họ có thể giải quyết. Bộ sưu tập thư viện tập trung vào dữ liệu phong phú của Spark (như MLlib) giúp các nhà khoa học dữ liệu dễ dàng vượt ra ngoài các vấn đề nằm gọn trên một máy duy nhất trong khi sử dụng nền tảng thống kê của họ. Trong khi đó, các kỹ sư sẽ học cách viết các chương trình phân tán đa năng trong Spark và vận hành các ứng dụng sản xuất. Các kỹ sư và nhà khoa học dữ liệu sẽ học được những chi tiết khác nhau từ cuốn sách này, nhưng cả hai đều có thể áp dụng Spark để giải quyết các vấn đề phân tán lớn trong lĩnh vực tương ứng của họ.

Các nhà khoa học dữ liệu tập trung vào việc trả lời các câu hỏi hoặc xây dựng mô hình từ dữ liệu. Họ thường có nền tảng về thống kê hoặc toán học và có một số hiểu biết về các công cụ như Python, R và SQL. Chúng tôi đã đảm bảo đưa Python và, nếu có liên quan, các ví dụ về SQL vào tất cả tài liệu của mình, cũng như tổng quan về máy học và thư viện trong Spark. Nếu bạn là một nhà khoa học dữ liệu, chúng tôi hy vọng rằng sau khi đọc cuốn sách này, bạn sẽ có thể sử dụng các phương pháp toán học tương tự để giải quyết các vấn đề, ngoại trừ việc nhanh hơn nhiều và ở quy mô lớn hơn nhiều.

Nhóm thứ hai mà cuốn sách này hướng đến là các kỹ sư phần mềm có một số kinh nghiệm với Java, Python hoặc ngôn ngữ lập trình khác. Nếu bạn là một kỹ sư, chúng tôi hy vọng rằng cuốn sách này sẽ chỉ cho bạn cách thiết lập cụm Spark, sử dụng shell Spark và viết các ứng dụng Spark để giải quyết các vấn đề xử lý song song. Nếu bạn quen thuộc với Hadoop, bạn sẽ có một chút khởi đầu thuận lợi để tìm ra cách tương tác với HDFS và cách quản lý cụm, nhưng dù bằng cách nào, chúng tôi sẽ đề cập đến các khái niệm thực thi phân tán cơ bản.

Bất kể bạn là nhà khoa học dữ liệu hay kỹ sư, để tận dụng tối đa cuốn sách này, bạn nên có một số hiểu biết về Python, Java, Scala hoặc ngôn ngữ tương tự. Chúng tôi cho rằng bạn đã có giải pháp lưu trữ cho dữ liệu của mình và chúng tôi sẽ đề cập đến cách tải và lưu dữ liệu từ nhiều giải pháp phổ biến, nhưng không đề cập đến cách thiết lập chúng. Nếu bạn không có kinh nghiệm với một trong những ngôn ngữ đó, đừng lo lắng: có những nguồn tài nguyên tuyệt vời để học những điều này. Chúng tôi sẽ giới thiệu một số cuốn sách có sẵn trong [“Sách hỗ trợ” ở trang xii](#).

Cuốn sách này được tổ chức như thế nào

Các chương của cuốn sách này được trình bày theo cách mà bạn có thể xem qua tài liệu từ đầu đến cuối. Vào đầu mỗi chương, chúng tôi sẽ đề cập đến những phần nào chúng tôi cho là có liên quan nhất đến các nhà khoa học dữ liệu và những phần nào chúng tôi cho là có liên quan nhất đến các kỹ sư. Điều đó nói lên rằng, chúng tôi hy vọng rằng tất cả tài liệu đều có thể tiếp cận được với độc giả ở bất kỳ trình độ nào.

Hai chương đầu tiên sẽ giúp bạn bắt đầu cài đặt Spark cơ bản trên máy tính xách tay của mình và cung cấp cho bạn ý tưởng về những gì bạn có thể thực hiện với Spark. Sau khi đã có động lực và thiết lập, chúng ta sẽ đi sâu vào Spark shell, một công cụ rất hữu ích để phát triển và tạo mẫu. Các chương tiếp theo sau đó sẽ đề cập chi tiết đến giao diện lập trình Spark, cách các ứng dụng thực thi trên cụm và các thư viện cấp cao hơn có sẵn trên Spark (như Spark SQL và MLlib).

Sách hỗ trợ

Nếu bạn là nhà khoa học dữ liệu và không có nhiều kinh nghiệm với Python, các cuốn sách *Học Python* và *Đầu tiên Python* (cả O'Reilly) đều là những lời giới thiệu tuyệt vời. Nếu

bạn có một số kinh nghiệm về Python và muốn nhiều hơn, *Lặn vào Python* (Apress) là một cuốn sách tuyệt vời giúp bạn hiểu sâu hơn về Python.

Nếu bạn là một kỹ sư và sau khi đọc cuốn sách này, bạn muốn mở rộng kỹ năng phân tích dữ liệu của mình, *Học máy cho tin tặc* và *Làm khoa học dữ liệu* là những cuốn sách tuyệt vời (cả hai đều của O'Reilly).

Cuốn sách này được thiết kế để người mới bắt đầu có thể tiếp cận. Chúng tôi dự định phát hành một bản tiếp theo chuyên sâu hơn cho những người muốn hiểu rõ hơn về nội dung bên trong của Spark.

Các quy ước được sử dụng trong cuốn sách này

Cuốn sách này sử dụng các quy ước đánh máy sau đây:

ngghiêng

Chỉ ra các thuật ngữ, URL, địa chỉ email, tên tệp và phần mở rộng tệp mới.

Chiều rộng không đổi

Được sử dụng cho danh sách chương trình cũng như trong các đoạn văn để tham chiếu đến các thành phần chương trình như tên biến hoặc hàm, cơ sở dữ liệu, kiểu dữ liệu, biến môi trường, câu lệnh và từ khóa.

Độ rộng không đổi đậm

Hiển thị các lệnh hoặc văn bản khác mà người dùng phải nhập theo đúng nghĩa đen.

Chiều rộng cố định nghiêng

Hiển thị văn bản cần được thay thế bằng giá trị do người dùng cung cấp hoặc bằng giá trị do ngữ cảnh xác định.



Yếu tố này biểu thị một lời khuyên hoặc gợi ý.



Yếu tố này biểu thị một cảnh báo hoặc thận trọng.

Ví dụ về mã

Tất cả các ví dụ mã được tìm thấy trong cuốn sách này đều có trên GitHub. Bạn có thể xem xét chúng và kiểm tra chúng từ <https://github.com/databricks/learning-spark>. Các ví dụ mã được cung cấp bằng Java, Scala và Python.



Các ví dụ Java của chúng tôi được viết để hoạt động với Java phiên bản 6 trở lên. Java 8 giới thiệu cú pháp mới có tên là *lambda* điều đó làm cho việc viết các hàm nội tuyến dễ dàng hơn nhiều, có thể đơn giản hóa mã Spark. Chúng tôi đã chọn không tận dụng cú pháp này trong hầu hết các ví dụ của mình, vì hầu hết các tổ chức vẫn chưa sử dụng Java 8. Nếu bạn muốn thử cú pháp Java 8, bạn có thể xem [bài đăng trên blog Databricks về chủ đề này](#). Một số ví dụ cũng sẽ được chuyển sang Java 8 và đăng lên trang GitHub của cuốn sách.

Cuốn sách này ở đây để giúp bạn hoàn thành công việc của mình. Nhìn chung, nếu mã ví dụ được cung cấp cùng với cuốn sách này, bạn có thể sử dụng nó trong các chương trình và tài liệu của mình. Bạn không cần phải liên hệ với chúng tôi để xin phép trừ khi bạn đang sao chép một phần đáng kể của mã. Ví dụ, việc viết một chương trình sử dụng một số đoạn mã từ cuốn sách này không yêu cầu phải xin phép. Việc bán hoặc phân phối một đĩa CD-ROM gồm các ví dụ từ sách O'Reilly thì cần phải xin phép. Trả lời một câu hỏi bằng cách trích dẫn cuốn sách này và trích dẫn mã ví dụ thì không cần xin phép. Việc kết hợp một lượng lớn mã ví dụ từ cuốn sách này vào tài liệu sản phẩm của bạn thì cần phải xin phép.

Chúng tôi đánh giá cao nhưng không yêu cầu ghi rõ nguồn. Ghi rõ nguồn thường bao gồm tiêu đề, tác giả, nhà xuất bản và ISBN. Ví dụ: "*Học tập Spark* của Holden Karau, Andy Konwinski, Patrick Wendell và Matei Zaharia (O'Reilly). Bản quyền 2015 Databricks, 978-1-449-35862-4."

Nếu bạn cảm thấy việc sử dụng các ví dụ mã của bạn nằm ngoài phạm vi sử dụng hợp lý hoặc quyền được nêu ở trên, vui lòng liên hệ với chúng tôi tại permission@oreilly.com.

Sách Safari® trực tuyến



Safari®

Sách Safari trực tuyến là một thư viện kỹ thuật số theo yêu cầu cung cấp chuyên **giao nội dung** dưới dạng sách và video từ các tác giả hàng đầu thế giới trong lĩnh vực công nghệ và kinh doanh.

Các chuyên gia công nghệ, nhà phát triển phần mềm, nhà thiết kế web, chuyên gia kinh doanh và sáng tạo sử dụng Safari Books Online làm nguồn tài nguyên chính cho nghiên cứu, giải quyết vấn đề, học tập và đào tạo cấp chứng chỉ.

Safari Books Online cung cấp một loạt các **kế hoạch và giá cả doanh nghiệp, chính phủ, giáo dục** và cá nhân.

Các thành viên có quyền truy cập vào hàng ngàn cuốn sách, video đào tạo và bản thảo trước khi xuất bản trong một cơ sở dữ liệu có thể tìm kiếm đầy đủ từ các nhà xuất bản như O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan

Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology và hàng trăm hơn. Để biết thêm thông tin về Safari Books Online, vui lòng truy cập trang web của chúng tôi [trực tuyến](#).

Làm thế nào để liên hệ với chúng tôi

Vui lòng gửi ý kiến và câu hỏi liên quan đến cuốn sách này tới nhà xuất bản:

Công ty truyền thông O'Reilly
1005 Đường cao tốc Gravenstein Bắc
Sebastopol, CA 95472
800-998-9938 (tại Hoa Kỳ hoặc Canada)
707-829-0515 (quốc tế hoặc địa phương)
707-829-0104 (fax)

Chúng tôi có một trang web cho cuốn sách này, nơi chúng tôi liệt kê các lỗi, ví dụ và bất kỳ thông tin bổ sung nào. Bạn có thể truy cập trang này tại <http://bit.ly/learning-spark>.

Để bình luận hoặc hỏi các câu hỏi kỹ thuật về cuốn sách này, hãy gửi email đến bookquestions@oreilly.com.

Để biết thêm thông tin về sách, khóa học, hội nghị và tin tức của chúng tôi, hãy xem trang web của chúng tôi tại <http://www.oreilly.com>.

Tìm chúng tôi trên Facebook: <http://facebook.com/oreilly> Theo dõi chúng

tôi trên Twitter: <http://twitter.com/oreillymedia> Xem chúng tôi trên

YouTube: <http://www.youtube.com/oreillymedia>

Lời cảm ơn

Các tác giả muốn cảm ơn những người đánh giá đã cung cấp phản hồi về cuốn sách này: Joseph Bradley, Dave Bridgeland, Chaz Chandler, Mick Davies, Sam DeHority, Vida Ha, Andrew Gal, Michael Gregson, Jan Joeppen, Stephan Jou, Jeff Martinez, Josh Mahonin, Andrew Or, Mike Patterson, Josh Rosen, Bruce Szalwinski, Xiangrui Meng và Reza Zadeh.

Các tác giả muốn gửi lời cảm ơn đặc biệt đến David Andrzejewski, David Butler, Juliet Hougland, Marek Kolodziej, Taka Shinagawa, Deborah Siegel, Tiến sĩ Normen Müller, Ali Ghodsi và Sameer Farooqui. Họ đã cung cấp phản hồi chi tiết về phần lớn các chương và giúp chỉ ra nhiều cải tiến đáng kể.

Chúng tôi cũng muốn cảm ơn các chuyên gia về chủ đề đã dành thời gian biên tập và viết một số phần trong chương của họ. Tathagata Das đã làm việc với chúng tôi theo một lịch trình rất chặt chẽ để hoàn thành Chương 10. Tathagata đã làm việc vượt quá mong đợi với việc làm rõ

ví dụ, trả lời nhiều câu hỏi và cải thiện luồng văn bản ngoài những đóng góp kỹ thuật của ông. Michael Armbrust đã giúp chúng tôi kiểm tra chương Spark SQL để đảm bảo tính chính xác. Joseph Bradley đã cung cấp ví dụ giới thiệu cho MLlib trong Chương 11. Reza Zadeh đã cung cấp các ví dụ về văn bản và mã để giảm chiều. Xiangrui Meng, Joseph Bradley và Reza Zadeh cũng cung cấp phản hồi về mặt kỹ thuật và biên tập cho chương MLlib.

Giới thiệu về Phân tích dữ liệu với Spark

Chương này cung cấp tổng quan cấp cao về Apache Spark là gì. Nếu bạn đã quen thuộc với Apache Spark và các thành phần của nó, hãy thoải mái chuyển sang [Chương 2](#).

Apache Spark là gì?

Apache Spark là một nền tảng điện toán cụm được thiết kế để *nhANH* và *mục đích chung*.

Về mặt tốc độ, Spark mở rộng mô hình MapReduce phổ biến để hỗ trợ hiệu quả nhiều loại tính toán hơn, bao gồm truy vấn tương tác và xử lý luồng. Tốc độ rất quan trọng trong việc xử lý các tập dữ liệu lớn, vì nó có nghĩa là sự khác biệt giữa việc khám phá dữ liệu một cách tương tác và chờ đợi nhiều phút hoặc nhiều giờ. Một trong những tính năng chính mà Spark cung cấp về tốc độ là khả năng chạy các phép tính trong bộ nhớ, nhưng hệ thống cũng hiệu quả hơn MapReduce đối với các ứng dụng phức tạp chạy trên đĩa.

Về mặt tổng quát, Spark được thiết kế để bao phủ một loạt khối lượng công việc mà trước đây yêu cầu các hệ thống phân tán riêng biệt, bao gồm các ứng dụng hàng loạt, thuật toán lặp, truy vấn tương tác và phát trực tuyến. Bằng cách hỗ trợ các khối lượng công việc này trong cùng một công cụ, Spark giúp dễ dàng và không tốn kém *để kết hợp* các loại xử lý khác nhau, thường cần thiết trong các quy trình phân tích dữ liệu sản xuất. Ngoài ra, nó làm giảm gánh nặng quản lý khi duy trì các công cụ riêng biệt.

Spark được thiết kế để có khả năng truy cập cao, cung cấp các API đơn giản trong Python, Java, Scala và SQL, cùng các thư viện tích hợp phong phú. Nó cũng tích hợp chặt chẽ với các công cụ Dữ liệu lớn khác. Đặc biệt, Spark có thể chạy trong các cụm Hadoop và truy cập bất kỳ nguồn dữ liệu Hadoop nào, bao gồm cả Cassandra.

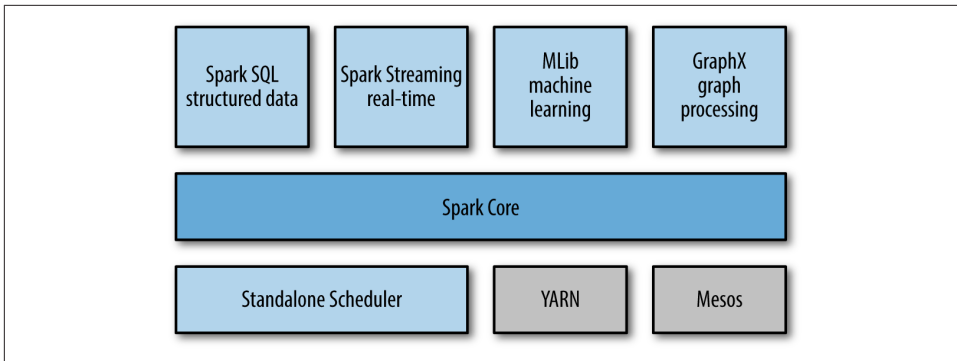
Một ngăn xếp thống nhất

Dự án Spark chứa nhiều thành phần tích hợp chặt chẽ. Về bản chất, Spark là một “công cụ tính toán” chịu trách nhiệm lập lịch, phân phối và giám sát các ứng dụng bao gồm nhiều tác vụ tính toán trên nhiều máy công nhân hoặc *cụm máy tính*. Vì engine cốt lõi của Spark vừa nhanh vừa đa năng, nên nó cung cấp năng lượng cho nhiều thành phần cấp cao hơn chuyên biệt cho nhiều khối lượng công việc khác nhau, chẳng hạn như SQL hoặc máy học. Các thành phần này được thiết kế để tương tác chặt chẽ, cho phép bạn kết hợp chúng như các thư viện trong một dự án phần mềm.

Triết lý tích hợp chặt chẽ có một số lợi ích. Đầu tiên, tất cả các thư viện và thành phần cấp cao hơn trong ngăn xếp đều được hưởng lợi từ các cải tiến ở các lớp thấp hơn. Ví dụ, khi công cụ cốt lõi của Spark thêm tối ưu hóa, các thư viện SQL và máy học cũng tự động tăng tốc. Thứ hai, chi phí liên quan đến việc chạy ngăn xếp được giảm thiểu, vì thay vì chạy 5–10 hệ thống phần mềm độc lập, một tổ chức chỉ cần chạy một hệ thống. Các chi phí này bao gồm triển khai, bảo trì, thử nghiệm, hỗ trợ và các chi phí khác. Điều này cũng có nghĩa là mỗi khi một thành phần mới được thêm vào ngăn xếp Spark, mọi tổ chức sử dụng Spark sẽ có thể dùng thử thành phần mới này ngay lập tức. Điều này thay đổi chi phí dùng thử một loại phân tích dữ liệu mới từ việc tải xuống, triển khai và tìm hiểu một dự án phần mềm mới cho đến việc nâng cấp Spark.

Cuối cùng, một trong những lợi thế lớn nhất của tích hợp chặt chẽ là khả năng xây dựng các ứng dụng kết hợp liền mạch các mô hình xử lý khác nhau. Ví dụ, trong Spark, bạn có thể viết một ứng dụng sử dụng máy học để phân loại dữ liệu theo thời gian thực khi dữ liệu được thu thập từ các nguồn phát trực tuyến. Đồng thời, các nhà phân tích có thể truy vấn dữ liệu kết quả, cũng theo thời gian thực, thông qua SQL (ví dụ: để liên kết dữ liệu với các tệp nhật ký không có cấu trúc). Ngoài ra, các kỹ sư dữ liệu và nhà khoa học dữ liệu tinh vi hơn có thể truy cập cùng một dữ liệu thông qua shell Python để phân tích ad hoc. Những người khác có thể truy cập dữ liệu trong các ứng dụng hàng loạt độc lập. Trong khi đó, nhóm CNTT chỉ phải duy trì một hệ thống.

Ở đây chúng tôi sẽ giới thiệu tóm tắt từng thành phần của Spark, được hiển thị trong [Hình 1-1](#).



Hình 1-1. Ngăn xếp Spark

Lõi tia lửa

Spark Core chứa các chức năng cơ bản của Spark, bao gồm các thành phần để lập lịch tác vụ, quản lý bộ nhớ, phục hồi lỗi, tương tác với hệ thống lưu trữ, v.v. Spark Core cũng là nơi chứa API xác định *bộ dữ liệu phân tán có khả năng phục hồi* (RDD), là sự trừu tượng hóa lập trình chính của Spark. RDD biểu thị một tập hợp các mục được phân bố trên nhiều nút tính toán có thể được thao tác song song. Spark Core cung cấp nhiều API để xây dựng và thao tác các tập hợp này.

Tia lửa SQL

Spark SQL là gói của Spark để làm việc với dữ liệu có cấu trúc. Nó cho phép truy vấn dữ liệu qua SQL cũng như biến thể Apache Hive của SQL—được gọi là Ngôn ngữ truy vấn Hive (HQL)—và nó hỗ trợ nhiều nguồn dữ liệu, bao gồm các bảng Hive, Parquet và JSON. Ngoài việc cung cấp giao diện SQL cho Spark, Spark SQL cho phép các nhà phát triển trộn lẫn các truy vấn SQL với các thao tác dữ liệu theo chương trình được hỗ trợ bởi RDD trong Python, Java và Scala, tất cả trong một ứng dụng duy nhất, do đó kết hợp SQL với các phân tích phức tạp. Sự tích hợp chặt chẽ này với môi trường điện toán phong phú do Spark cung cấp khiến Spark SQL không giống bất kỳ công cụ kho dữ liệu nguồn mở nào khác. Spark SQL đã được thêm vào Spark ở phiên bản 1.0.

Shark là một dự án SQL-on-Spark cũ hơn của Đại học California, Berkeley, đã sửa đổi Apache Hive để chạy trên Spark. Hiện tại, nó đã được thay thế bằng Spark SQL để cung cấp khả năng tích hợp tốt hơn với Spark engine và API ngôn ngữ.

Spark phát trực tuyến

Spark Streaming là một thành phần Spark cho phép xử lý luồng dữ liệu trực tiếp. Ví dụ về luồng dữ liệu bao gồm các tệp nhật ký được tạo bởi máy chủ web sản xuất hoặc hàng đợi tin nhắn chứa các bản cập nhật trạng thái do người dùng của dịch vụ web đăng. Spark

Streaming cung cấp một API để thao tác các luồng dữ liệu gần giống với API RDD của Spark Core, giúp các lập trình viên dễ dàng tìm hiểu dự án và chuyển đổi giữa các ứng dụng thao tác dữ liệu được lưu trữ trong bộ nhớ, trên đĩa hoặc đến theo thời gian thực. Bên dưới API của mình, Spark Streaming được thiết kế để cung cấp cùng mức độ chịu lỗi, thông lượng và khả năng mở rộng như Spark Core.

Thư viện MLlib

Spark đi kèm với một thư viện chứa chức năng học máy (ML) phổ biến, được gọi là MLlib. MLlib cung cấp nhiều loại thuật toán học máy, bao gồm phân loại, hồi quy, phân cụm và lọc cộng tác, cũng như hỗ trợ các chức năng như đánh giá mô hình và nhập dữ liệu. Nó cũng cung cấp một số nguyên hàm ML cấp thấp hơn, bao gồm thuật toán tối ưu hóa gradient descent chung. Tất cả các phương pháp này được thiết kế để mở rộng trên một cụm.

Đồ thịX

GraphX là một thư viện để thao tác đồ thị (ví dụ: đồ thị bạn bè của mạng xã hội) và thực hiện các phép tính song song đồ thị. Giống như Spark Streaming và Spark SQL, GraphX mở rộng API Spark RDD, cho phép chúng ta tạo đồ thị có hướng với các thuộc tính tùy ý được gắn vào mỗi đỉnh và cạnh. GraphX cũng cung cấp nhiều toán tử khác nhau để thao tác đồ thị (ví dụ: đồ thị con và mapVertices) và một thư viện các thuật toán đồ thị phổ biến (ví dụ: PageRank và đếm tam giác).

Quản lý cụm

Về cơ bản, Spark được thiết kế để mở rộng hiệu quả từ một đến hàng nghìn nút tính toán. Để đạt được điều này trong khi tối đa hóa tính linh hoạt, Spark có thể chạy trên nhiều *quản lý cụm*, bao gồm Hadoop YARN, Apache Mesos và một trình quản lý cụm đơn giản có trong Spark được gọi là Standalone Scheduler. Nếu bạn chỉ cài đặt Spark trên một tập hợp máy trống, Standalone Scheduler cung cấp một cách dễ dàng để bắt đầu; tuy nhiên, nếu bạn đã có cụm Hadoop YARN hoặc Mesos, hỗ trợ của Spark cho các trình quản lý cụm này cho phép các ứng dụng của bạn cũng chạy trên chúng. **Chương 7** khám phá các tùy chọn khác nhau và cách chọn trình quản lý cụm phù hợp.

Ai sử dụng Spark và để làm gì?

Bởi vì Spark là một khuôn khổ mục đích chung cho điện toán cụm, nó được sử dụng cho nhiều ứng dụng khác nhau. Trong **Lời nói đầu** chúng tôi đã phác thảo hai nhóm độc giả mà cuốn sách này hướng đến: các nhà khoa học dữ liệu và kỹ sư. Chúng ta hãy xem xét kỹ hơn từng nhóm và cách họ sử dụng Spark. Không có gì ngạc nhiên khi các trường hợp sử dụng điển hình khác nhau giữa hai nhóm,

nhưng chúng ta có thể phân loại chúng thành hai loại, *khoa học dữ liệu* và *ứng dụng dữ liệu*.

Tất nhiên, đây là những ngành học và mô hình sử dụng không chính xác, và nhiều người có kỹ năng từ cả hai, đôi khi đóng vai trò là nhà khoa học dữ liệu đang điều tra, sau đó "đổi mũ" và viết một ứng dụng xử lý dữ liệu đã được củng cố. Tuy nhiên, việc xem xét riêng hai nhóm và các trường hợp sử dụng tương ứng của chúng có thể giúp làm sáng tỏ vấn đề.

Nhiệm vụ khoa học dữ liệu

Khoa học dữ liệu, một ngành học mới nổi trong vài năm trở lại đây, tập trung vào việc phân tích dữ liệu. Mặc dù không có định nghĩa chuẩn, nhưng đối với mục đích của chúng tôi, *nhà khoa học dữ liệu* là người có nhiệm vụ chính là phân tích và mô hình hóa dữ liệu. Các nhà khoa học dữ liệu có thể có kinh nghiệm về SQL, thống kê, mô hình dự đoán (học máy) và lập trình, thường là bằng Python, Matlab hoặc R. Các nhà khoa học dữ liệu cũng có kinh nghiệm về các kỹ thuật cần thiết để chuyển đổi dữ liệu thành các định dạng có thể phân tích để có được thông tin chi tiết (đôi khi được gọi là *tranh giành dữ liệu*).

Các nhà khoa học dữ liệu sử dụng các kỹ năng của họ để phân tích dữ liệu với mục tiêu trả lời một câu hỏi hoặc khám phá ra những hiểu biết sâu sắc. Thông thường, quy trình làm việc của họ liên quan đến phân tích ad hoc, vì vậy họ sử dụng các shell tương tác (so với việc xây dựng các ứng dụng phức tạp) cho phép họ xem kết quả của các truy vấn và đoạn mã trong thời gian ngắn nhất. Tốc độ và API đơn giản của Spark tỏa sáng cho mục đích này và các thư viện tích hợp của nó có nghĩa là nhiều thuật toán có sẵn ngay lập tức.

Spark hỗ trợ các tác vụ khác nhau của khoa học dữ liệu với một số thành phần. Spark shell giúp dễ dàng thực hiện phân tích dữ liệu tương tác bằng Python hoặc Scala. Spark SQL cũng có một SQL shell riêng có thể được sử dụng để thực hiện khám phá dữ liệu bằng SQL hoặc Spark SQL có thể được sử dụng như một phần của chương trình Spark thông thường hoặc trong Spark shell. Học máy và phân tích dữ liệu được hỗ trợ thông qua các thư viện MLlib. Ngoài ra, còn có hỗ trợ gọi ra các chương trình bên ngoài trong Matlab hoặc R. Spark cho phép các nhà khoa học dữ liệu giải quyết các vấn đề với kích thước dữ liệu lớn hơn so với trước đây họ có thể làm bằng các công cụ như R hoặc Pandas.

Đôi khi, sau giai đoạn khám phá ban đầu, công việc của nhà khoa học dữ liệu sẽ được "sản xuất hóa" hoặc mở rộng, được củng cố (tức là được tạo ra để có khả năng chịu lỗi) và được điều chỉnh để trở thành ứng dụng xử lý dữ liệu sản xuất, bản thân nó là một thành phần của ứng dụng kinh doanh. Ví dụ, cuộc điều tra ban đầu của nhà khoa học dữ liệu có thể dẫn đến việc tạo ra hệ thống đề xuất sản xuất được tích hợp vào ứng dụng web và được sử dụng để tạo ra các đề xuất sản phẩm cho người dùng. Thông thường, một người hoặc nhóm khác sẽ dẫn đầu quá trình sản xuất công việc của nhà khoa học dữ liệu và người đó thường là một kỹ sư.

Ứng dụng xử lý dữ liệu

Trường hợp sử dụng chính khác của Spark có thể được mô tả trong bối cảnh của cá nhân kỹ sư. Đối với mục đích của chúng tôi ở đây, chúng tôi coi kỹ sư là một nhóm lớn các nhà phát triển phần mềm sử dụng Spark để xây dựng các ứng dụng xử lý dữ liệu sản xuất. Những nhà phát triển này thường hiểu biết về các nguyên tắc của kỹ thuật phần mềm, chẳng hạn như đóng gói, thiết kế giao diện và lập trình hướng đối tượng. Họ thường có bằng về khoa học máy tính. Họ sử dụng các kỹ năng kỹ thuật của mình để thiết kế và xây dựng các hệ thống phần mềm triển khai trường hợp sử dụng kinh doanh.

Đối với các kỹ sư, Spark cung cấp một cách đơn giản để song song hóa các ứng dụng này trên các cụm và ẩn đi sự phức tạp của lập trình hệ thống phân tán, giao tiếp mạng và khả năng chịu lỗi. Hệ thống cung cấp cho họ đủ quyền kiểm soát để giám sát, kiểm tra và điều chỉnh các ứng dụng trong khi cho phép họ triển khai các tác vụ chung một cách nhanh chóng. Bản chất mô-đun của API (dựa trên việc truyền các bộ sưu tập đối tượng phân tán) giúp dễ dàng đưa công việc vào các thư viện có thể tái sử dụng và kiểm tra cục bộ.

Người dùng Spark chọn sử dụng nó cho các ứng dụng xử lý dữ liệu của họ vì nó cung cấp nhiều chức năng, dễ học và sử dụng, đồng thời cũng hoàn thiện và đáng tin cậy.

Một Lịch Sử Ngắn Gọn Của Spark

Spark là một dự án nguồn mở được xây dựng và duy trì bởi một cộng đồng các nhà phát triển đa dạng và phát triển mạnh mẽ. Nếu bạn hoặc tổ chức của bạn đang dùng thử Spark lần đầu tiên, bạn có thể quan tâm đến lịch sử của dự án. Spark bắt đầu vào năm 2009 như một dự án nghiên cứu tại Phòng thí nghiệm RAD của UC Berkeley, sau này trở thành AMPLab. Các nhà nghiên cứu trong phòng thí nghiệm trước đây đã làm việc trên Hadoop Map-Reduce và nhận thấy rằng MapReduce không hiệu quả đối với các công việc tính toán tương tác và lặp lại. Do đó, ngay từ đầu, Spark đã được thiết kế để nhanh chóng cho các truy vấn tương tác và thuật toán lặp lại, đưa ra các ý tưởng như hỗ trợ lưu trữ trong bộ nhớ và phục hồi lỗi hiệu quả.

Các bài báo nghiên cứu về Spark đã được công bố tại các hội nghị học thuật và ngay sau khi ra đời vào năm 2009, nó đã nhanh hơn MapReduce từ 10–20 lần đối với một số công việc nhất định.

Một số người dùng đầu tiên của Spark là các nhóm khác bên trong UC Berkeley, bao gồm các nhà nghiên cứu về máy học như dự án Mobile Millennium, đã sử dụng Spark để theo dõi và dự đoán tình trạng tắc nghẽn giao thông ở Khu vực Vịnh San Francisco. Tuy nhiên, trong một thời gian rất ngắn, nhiều tổ chức bên ngoài đã bắt đầu sử dụng Spark và hiện nay, hơn 50 tổ chức đã tự liệt kê mình trên [Trang Spark PoweredBy](#) và hàng chục người nói về các trường hợp sử dụng của họ tại các sự kiện cộng đồng Spark như [Gặp gỡ Spark](#) và [Hội nghị thượng đỉnh Spark](#). Ngoài UC Berkeley, những đơn vị đóng góp chính cho Spark bao gồm Databricks, Yahoo! và Intel.

Vào năm 2011, AMPLab bắt đầu phát triển các thành phần cấp cao hơn trên Spark, chẳng hạn như Shark (Hive trên Spark)¹ và Spark Streaming. Những thành phần này và các thành phần khác đôi khi được gọi là **Berkeley Data Analytics Stack (BDAS)**.

Spark lần đầu tiên được mở mã nguồn vào tháng 3 năm 2010 và được chuyển giao cho Apache Software Foundation vào tháng 6 năm 2013, nơi hiện là một dự án cấp cao nhất.

Phiên bản và bản phát hành Spark

Kể từ khi thành lập, Spark đã là một dự án và cộng đồng rất năng động, với số lượng người đóng góp tăng lên sau mỗi lần phát hành. Spark 1.0 có hơn 100 người đóng góp cá nhân. Mặc dù mức độ hoạt động đã tăng nhanh chóng, cộng đồng vẫn tiếp tục phát hành các phiên bản cập nhật của Spark theo lịch trình thường xuyên. Spark 1.0 được phát hành vào tháng 5 năm 2014. Cuốn sách này tập trung chủ yếu vào Spark 1.1.0 trở lên, mặc dù hầu hết các khái niệm và ví dụ cũng hoạt động trong các phiên bản trước đó.

Các lớp lưu trữ cho Spark

Spark có thể tạo các tập dữ liệu phân tán từ bất kỳ tệp nào được lưu trữ trong hệ thống tệp phân tán Hadoop (HDFS) hoặc các hệ thống lưu trữ khác được API Hadoop hỗ trợ (bao gồm hệ thống tệp cục bộ của bạn, Amazon S3, Cassandra, Hive, HBase, v.v.). Điều quan trọng cần nhớ là Spark không yêu cầu Hadoop; nó chỉ hỗ trợ các hệ thống lưu trữ triển khai API Hadoop. Spark hỗ trợ các tệp văn bản, SequenceFiles, Avro, Parquet và bất kỳ Hadoop InputFormat nào khác. Chúng ta sẽ xem xét cách tương tác với các nguồn dữ liệu này trong **Chương 5**.

¹ Shark đã được thay thế bằng Spark SQL.

Tải xuống Spark và Bắt đầu

Trong chương này, chúng ta sẽ hướng dẫn quy trình tải xuống và chạy Spark ở chế độ cục bộ trên một máy tính. Chương này được viết cho bất kỳ ai mới làm quen với Spark, bao gồm cả nhà khoa học dữ liệu và kỹ sư.

Spark có thể được sử dụng từ Python, Java hoặc Scala. Để hưởng lợi từ cuốn sách này, bạn không cần phải là một lập trình viên chuyên nghiệp, nhưng chúng tôi cho rằng bạn cảm thấy thoải mái với cú pháp cơ bản của ít nhất một trong những ngôn ngữ này. Chúng tôi sẽ đưa ra các ví dụ trong tất cả các ngôn ngữ bất cứ khi nào có thể.

Bản thân Spark được viết bằng Scala và chạy trên Java Virtual Machine (JVM). Để chạy Spark trên máy tính xách tay hoặc cụm máy tính, tất cả những gì bạn cần là cài đặt Java 6 trở lên. Nếu bạn muốn sử dụng Python API, bạn cũng sẽ cần trình thông dịch Python (phiên bản 2.6 trở lên). Spark hiện chưa hoạt động với Python 3.

Đang tải Spark

Bước đầu tiên để sử dụng Spark là tải xuống và giải nén nó. Hãy bắt đầu bằng cách tải xuống phiên bản Spark đã biên dịch trước gần đây. Truy cập <http://spark.apache.org/download.html>, chọn loại gói “Được xây dựng sẵn cho Hadoop 2.4 trở lên” và nhấp vào “Tải xuống trực tiếp”. Thao tác này sẽ tải xuống tệp TAR đã nén hoặc *tarball*, gọi điện *spark-1.2.0-bin-hadoop2.4.tgz*.



Người dùng Windows có thể gặp sự cố khi cài đặt Spark vào thư mục có khoảng trắng trong tên. Thay vào đó, hãy cài đặt Spark trong thư mục không có khoảng trắng (ví dụ: *C:\tia lửa*).

Bạn không cần phải có Hadoop, nhưng nếu bạn có cụm Hadoop hoặc cài đặt HDFS hiện có, hãy tải xuống phiên bản phù hợp. Bạn có thể thực hiện việc này từ <http://spark.apache.org/downloads.html> bằng cách chọn một loại gói khác, nhưng chúng sẽ có tên tệp hơi khác nhau. Xây dựng từ nguồn cũng có thể thực hiện được; bạn có thể tìm thấy mã nguồn mới nhất trên [GitHub](#) hoặc chọn loại gói “Mã nguồn” khi tải xuống.



Hầu hết các biến thể Unix và Linux, bao gồm cả Mac OS X, đều đi kèm với một công cụ dòng lệnh được gọi là `hắc ín` có thể được sử dụng để giải nén các tệp TAR. Nếu hệ điều hành của bạn không có `hắc ín` đã cài đặt, hãy thử tìm kiếm trên Internet một trình giải nén TAR miễn phí—ví dụ, trên Windows, bạn có thể muốn thử 7-Zip.

Bây giờ chúng ta đã tải xuống Spark, hãy giải nén nó và xem những gì đi kèm với bản phân phối Spark mặc định. Để làm điều đó, hãy mở một thiết bị đầu cuối, chuyển đến thư mục mà bạn đã tải xuống Spark và giải nén tệp. Thao tác này sẽ tạo một thư mục mới có cùng tên nhưng không có tệp cuối cùng `.tgz` hậu tố. Chuyển vào thư mục đó và xem những gì bên trong. Bạn có thể sử dụng các lệnh sau để thực hiện tất cả những điều đó:

```
địa CD~  
tar -xf spark-1.2.0-bin-hadoop2.4.tgz  
địa CDspark-1.2.0-bin-hadoop2.4 ls
```

Trong dòng chữ `hắc ín` lệnh, các `xlá` cờ cho biết `hắc ín` chúng tôi đang trích xuất các tệp tin và nếu `hắc ín` chỉ định tên của tarball. `l` là lệnh liệt kê nội dung của thư mục Spark. Chúng ta hãy xem xét sơ qua tên và mục đích của một số tệp và thư mục quan trọng hơn mà bạn thấy ở đây đi kèm với Spark:

README.md

Bao gồm hướng dẫn ngắn gọn để bắt đầu sử dụng Spark.

thùng rác

Chứa các tệp thực thi có thể được sử dụng để tương tác với Spark theo nhiều cách khác nhau (ví dụ: shell Spark, chúng ta sẽ đề cập sau trong chương này).

lỗi, phát trực tuyến, python, ...

Chứa mã nguồn của các thành phần chính của dự án Spark.

ví dụ

Bao gồm một số tác vụ độc lập hữu ích của Spark mà bạn có thể xem và chạy để tìm hiểu về Spark API.

Đừng lo lắng về số lượng lớn các thư mục và tệp mà dự án Spark đi kèm; chúng tôi sẽ đề cập đến hầu hết những điều này trong phần còn lại của cuốn sách này. Bây giờ, chúng ta hãy bắt đầu và thử nghiệm các shell Python và Scala của Spark. Chúng ta sẽ bắt đầu bằng cách chạy một số

các ví dụ đi kèm với Spark. Sau đó, chúng ta sẽ viết, biên dịch và chạy một tác vụ Spark đơn giản của riêng mình.

Tất cả công việc chúng ta sẽ làm trong chương này sẽ được thực hiện với Spark đang chạy trong *chế độ cục bộ*; tức là chế độ không phân phối, chỉ sử dụng một máy duy nhất. Spark có thể chạy trong nhiều chế độ hoặc môi trường khác nhau. Ngoài chế độ cục bộ, Spark cũng có thể chạy trên Mesos, YARN hoặc Standalone Scheduler có trong bản phân phối Spark. Chúng tôi sẽ đề cập chi tiết đến các chế độ triển khai khác nhau trong [Chương 7](#).

Giới thiệu về Python và Scala Shell của Spark

Spark đi kèm với các shell tương tác cho phép phân tích dữ liệu tùy ý. Các shell của Spark sẽ quen thuộc nếu bạn đã sử dụng các shell khác như trong R, Python và Scala hoặc các shell hệ điều hành như Bash hoặc dấu nhắc lệnh Windows.

Tuy nhiên, không giống như hầu hết các shell khác cho phép bạn thao tác dữ liệu bằng đĩa và bộ nhớ trên một máy duy nhất, shell của Spark cho phép bạn tương tác với dữ liệu được phân phối trên đĩa hoặc trong bộ nhớ trên nhiều máy và Spark sẽ tự động phân phối quá trình xử lý này.

Vì Spark có thể tải dữ liệu vào bộ nhớ trên các nút worker, nhiều phép tính phân tán, thậm chí là những phép tính xử lý hàng terabyte dữ liệu trên hàng chục máy, có thể chạy trong vài giây. Điều này làm cho loại phân tích lặp lại, tùy ý và khám phá thường được thực hiện trong shell trở nên phù hợp với Spark. Spark cung cấp cả shell Python và Scala đã được tăng cường để hỗ trợ kết nối với cụm.



Hầu hết cuốn sách này bao gồm mã trong tất cả các ngôn ngữ của Spark, nhưng shell tương tác chỉ khả dụng trong Python và Scala. Vì shell rất hữu ích để học API, chúng tôi khuyên bạn nên sử dụng một trong những ngôn ngữ này cho các ví dụ này ngay cả khi bạn là nhà phát triển Java. API tương tự nhau trong mọi ngôn ngữ.

Cách dễ nhất để chứng minh sức mạnh của shell Spark là bắt đầu sử dụng một trong số chúng để phân tích dữ liệu đơn giản. Hãy cùng xem qua ví dụ từ [Hướng dẫn bắt đầu nhanh](#) trong tài liệu chính thức của Spark.

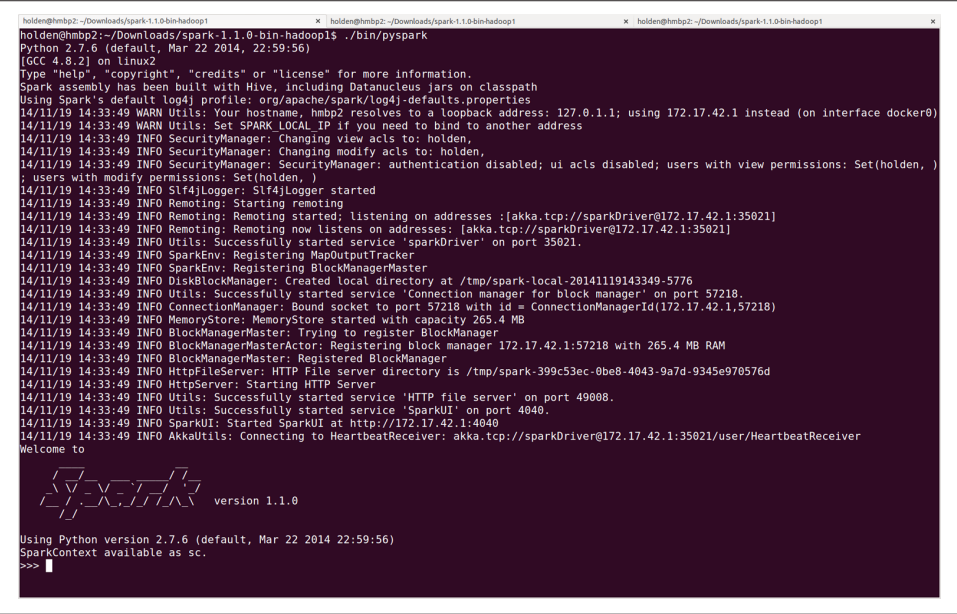
Bước đầu tiên là mở một trong các shell của Spark. Để mở phiên bản Python của shell Spark, mà chúng tôi cũng gọi là PySpark Shell, hãy vào thư mục Spark của bạn và nhập:

```
thùng rác/pyspark
```

(Hoặc `bin\pyspark` trong Windows.) Để mở phiên bản Scala của shell, hãy nhập:

```
thùng rác/vô tia lửa
```

Dấu nhắc shell sẽ xuất hiện trong vòng vài giây. Khi shell khởi động, bạn sẽ thấy rất nhiều thông báo nhật ký. Bạn có thể cần nhấn Enter một lần để xóa đầu ra nhật ký và đến dấu nhắc shell. **Hình 2-1** hiển thị giao diện của PySpark khi bạn mở nó.



Hình 2-1. Shell PySpark với đầu ra ghi nhật ký mặc định

Bạn có thể thấy các câu lệnh ghi nhật ký được in trong shell gây mất tập trung. Bạn có thể kiểm soát mức độ chi tiết của việc ghi nhật ký. Để thực hiện việc này, bạn có thể tạo một tệp trong *conf* thư mục được gọi là *log4j.thuộc tính*. Các nhà phát triển Spark đã bao gồm một mẫu cho tệp này được gọi là *log4j.properties.mẫu*. Để làm cho việc ghi nhật ký ít chi tiết hơn, hãy tạo một bản sao của *conf/log4j.properties.mẫu* gọi điện *conf/log4j.thuộc tính* và tìm dòng sau:

log4j.rootLogger=THÔNG TIN, bảng điều khiển

Sau đó hạ thấp mức nhật ký để chúng ta chỉ hiển thị các thông báo CẢNH BÁO và ở trên bằng cách thay đổi thành như sau:

log4j.rootLogger=CẢNH BÁO, bảng điều khiển

Khi bạn mở lại shell, bạn sẽ thấy ít đầu ra hơn (**Hình 2-2**).

```
holden@hmbp2: ~/Downloads/spark-1.1.0-bin-hadoop1
holden@hmbp2:~/Downloads/spark-1.1.0-bin-hadoop1$ ./bin/pyspark
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license()" for more information.
Spark assembly has been built with Hive, including Datanucleus jars on classpath
14/11/19 14:38:03 WARN Utils: Your hostname, hmbp2 resolves to a loopback address: 127.0.1.1; using 172.17.42.1 instead (on interface docker0)
14/11/19 14:38:03 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Welcome to

  ____      _
 / ___|    / \
| |  | |  / _ \
| |  | | / ___ \
| |  | |/_/   \_\
|_|  |_____/___)
version 1.1.0

Using Python version 2.7.6 (default, Mar 22 2014 22:59:56)
SparkContext available as sc.
>>>
```

Hình 2-2. Vở PySpark với đầu ra ghi nhật ký ít hơn



Sử dụng IPython

IPython là một shell Python nâng cao mà nhiều người dùng Python ưa thích, cung cấp các tính năng như hoàn thành tab. Bạn có thể tìm thấy hướng dẫn cài đặt tại <http://ipython.org>. Bạn có thể sử dụng IPython với Spark bằng cách thiết lập IPYTHON biến môi trường thành 1:

```
IPYTHON=1./bin/pyspark
```

Để sử dụng IPython Notebook, một phiên bản IPython dựa trên trình duyệt web, hãy sử dụng:

```
IPYTHON_OPTS="sổ tay"./bin/pyspark
```

Trên Windows, hãy đặt biến và chạy shell như sau:

```
bộIPYTHON=1
bin\pyspark
```

Trong Spark, chúng tôi thể hiện tính toán của mình thông qua các hoạt động trên các bộ sưu tập phân tán được tự động song song hóa trên toàn bộ cụm. Các bộ sưu tập này được gọi là *bộ dữ liệu phân tán có khả năng phục hồi* hoặc RDD. RDD là sự trừu tượng cơ bản của Spark dành cho dữ liệu phân tán và tính toán.

Trước khi chúng ta nói thêm về RDD, hãy tạo một RDD trong shell từ tệp văn bản cục bộ và thực hiện một số phân tích ad hoc rất đơn giản bằng cách làm theo [Ví dụ 2-1](#) cho Python hoặc [Ví dụ 2-2](#) cho Scala.

Ví dụ 2-1. Đếm dòng Python

```
>>> dòng=sc.Tập văn bản("README.md")#Tạo một RDD có tên là dòng

>>> dòng.đếm()#Đếm số lượng mục trong RDD này127

>>> dòng.Đầu tiên()#Mục đầu tiên trong RDD này, tức là dòng đầu tiên của README.md
bạn'# Apache Spark'
```

Ví dụ 2-2. Đếm dòng Scala

```
thang đo>giá trịdòng=sc.Tập văn bản("README.md")//Tạo một RDD có tên là dòng
dòng:tia lửa.RDD[Sợi dây]=Bản đồRDD[...]

thang đo>dòng.đếm()//Đếm số lượng mục trong RDD nàyđộ phân
giải0:Dài=127

thang đo>dòng.Đầu tiên()//Mục đầu tiên trong RDD này, tức là dòng đầu tiên của README.md độ phân
giải1:Sợi dây=#Tia lửa Apache
```

Để thoát khỏi bất kỳ shell nào, hãy nhấn Ctrl-D.



Chúng ta sẽ thảo luận thêm về nó trong **Chương 7**, nhưng một trong những tin nhắn bạn có thể đã nhận thấy là **THÔNG TIN SparkUI**: Bắt đầu SparkUI lúc [http://\[địa chỉ IP\]:4040](http://[địa chỉ IP]:4040). Bạn có thể truy cập Spark UI tại đó và xem đủ loại thông tin về tác vụ và cụm của mình.

Trong Ví dụ 2-1 và 2-2, biến được gọi là `dòng` là một RDD, được tạo ra ở đây từ một tập văn bản trên máy cục bộ của chúng tôi. Chúng ta có thể chạy nhiều hoạt động song song khác nhau trên RDD, chẳng hạn như đếm số phần tử trong tập dữ liệu (ở đây là các dòng văn bản trong tập) hoặc in phần tử đầu tiên. Chúng ta sẽ thảo luận sâu hơn về RDD trong các chương sau, nhưng trước khi đi sâu hơn, hãy dành một chút thời gian để giới thiệu các khái niệm cơ bản về Spark.

Giới thiệu về các khái niệm Core Spark

Bây giờ bạn đã chạy mã Spark đầu tiên bằng shell, đã đến lúc tìm hiểu chi tiết hơn về lập trình bằng shell.

Ở cấp độ cao, mọi ứng dụng Spark đều bao gồm một *chương trình điều khiển* khởi chạy nhiều hoạt động song song khác nhau trên một cụm. Chương trình điều khiển chứa ứng dụng của bạn chủ yếu chức năng và định nghĩa các tập dữ liệu phân tán trên cụm, sau đó áp dụng các hoạt động cho chúng. Trong các ví dụ trước, chương trình điều khiển chính là Spark shell và bạn chỉ cần nhập các hoạt động bạn muốn chạy.

Các chương trình điều khiển truy cập Spark thông qua một `SparkContext` đối tượng, biểu diễn kết nối đến cụm máy tính. Trong shell, `SparkContext` tự động

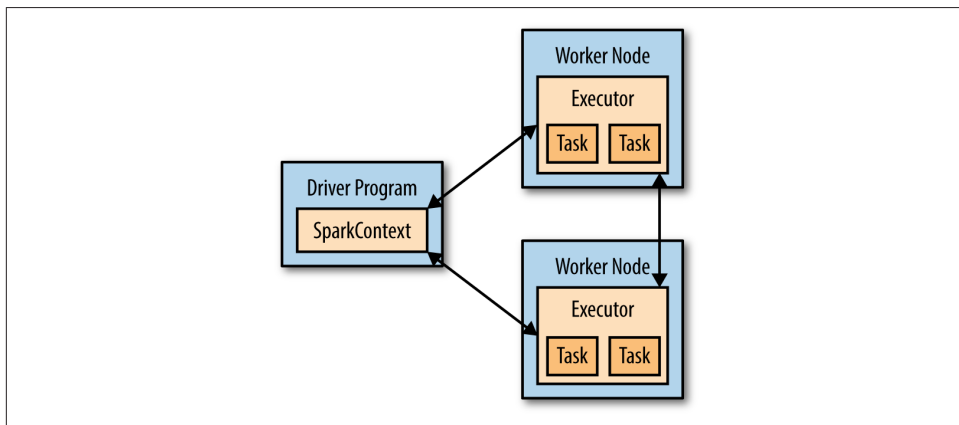
được tạo cho bạn như biến được gọi là `sc`. Hãy thử in ra để xem loại của nó, như được hiển thị trong Ví dụ 2-3.

Ví dụ 2-3. Kiểm tra biến `sc`

```
>>> sc
<công viên pyspark.bồi cảnh.SparkContextsự vật Tại 0x1025b8f90>
```

Khi bạn có `SparkContext`, bạn có thể sử dụng nó để xây dựng RDD. Trong Ví dụ 2-1 và 2-2, chúng tôi đã gọi `sc.textFile()` để tạo RDD biểu diễn các dòng văn bản trong một tệp. Sau đó, chúng ta có thể chạy nhiều thao tác khác nhau trên các dòng này, chẳng hạn như `sc.collect()`.

Để chạy các hoạt động này, các chương trình điều khiển thường quản lý một số nút được gọi là *người thi hành di chúc*. Ví dụ, nếu chúng ta đang chạy `sc.collect()` hoạt động trên một cụm, các máy khác nhau có thể đếm các dòng trong các phạm vi khác nhau của tệp. Vì chúng tôi chỉ chạy Spark shell cục bộ, nên nó thực hiện tất cả công việc của mình trên một máy duy nhất—nhưng bạn có thể kết nối cùng một shell với một cụm để phân tích dữ liệu song song. Hình 2-3 hiển thị cách Spark thực thi trên một cụm.



Hình 2-3. Các thành phần để thực thi phân tán trong Spark

Cuối cùng, rất nhiều API của Spark xoay quanh việc truyền các hàm cho các toán tử của nó để chạy chúng trên cụm. Ví dụ, chúng ta có thể mở rộng ví dụ README của mình bằng cách lọc các dòng trong tệp có chứa một từ, chẳng hạn như *Trần*, như thể hiện trong Ví dụ 2-4 (cho Python) và Ví dụ 2-5 (dành cho Scala).

Ví dụ 2-4. Ví dụ lọc Python

```
>>> dòng=sc.Tệp văn bản("README.md")
>>> Dòng python=dòng.lọc(lambda đường kẻ:"Trần" TRONG đường kẻ)
```

```
>>> Dòng python.Đầu tiên() u'##  
Vỏ Python tương tác'
```

Ví dụ 2-5. Ví dụ về lọc Scala

```
thang đo>giá trịdòng=sc.Tập văn bản("README.md")//Tạo một RDD có tên là dòng  
dòng:tia lửa.RDD[Sợi dây]=Bản đồRDD[...]
```

```
thang đo>giá trịDòng python=dòng.lọc(đường kẻ=>đường kẻ.chứa("Trăn"))  
Dòng python:tia lửa.RDD[Sợi dây]=LọcRDD[...]
```

```
thang đo>Dòng python.Đầu tiên()  
độ phân giải0:Sợi dây= ##Vỏ Python tương tác
```

Truyền hàm cho Spark

Nếu bạn không quen thuộc với `lambda` hoặc `=>` cú pháp trong Ví dụ 2-4 và 2-5, đây là cách viết tắt để định nghĩa hàm nội tuyến trong Python và Scala. Khi sử dụng Spark trong các ngôn ngữ này, bạn cũng có thể định nghĩa hàm riêng biệt rồi truyền tên hàm đó cho Spark. Ví dụ, trong Python:

```
định nghĩa có Python(đường kẻ):  
trở lại "Trăn" TRONG đường kẻ
```

```
Dòng python=dòng.lọc(có Python)
```

Việc truyền các hàm cho Spark cũng có thể thực hiện được trong Java, nhưng trong trường hợp này, chúng được định nghĩa là các lớp, triển khai một giao diện được gọi là `Chức năng`. Ví dụ:

```
JavaRDD<Sợi dây>Dòng python=dòng.lọc(  
    mới Chức năng<Sợi dây, Boolean>() {  
        Boolean gọi (Dòng dây) {trở lại đường kẻ.chứa("Trăn"); }  
    }  
);
```

Java 8 giới thiệu cú pháp viết tắt được gọi là `lambda` trông giống Python và Scala. Đây là cách mã sẽ trông như thế nào với cú pháp này:

```
JavaRDD<Sợi dây>Dòng python=dòng.lọc(đường kẻ->đường kẻ.chứa("Trăn"));
```

Chúng tôi thảo luận thêm về việc truyền các hàm trong "Truyền hàm cho Spark" ở trang 30.

Mặc dù chúng tôi sẽ đề cập chi tiết hơn về Spark API sau, nhưng phần lớn sự kỳ diệu của nó là các hoạt động dựa trên hàm như `lọc` mà còn song song hóa trên toàn cụm. Nghĩa là Spark tự động lấy hàm của bạn (ví dụ: `dòng.chứa("Python")`) và chuyển nó đến các nút thực thi. Do đó, bạn có thể viết mã trong một chương trình trình điều khiển duy nhất và tự động chạy các phần của nó trên nhiều nút. Chương 3 bao gồm chi tiết về API RDD.

Ứng dụng độc lập

Phần cuối cùng còn thiếu trong chuyến tham quan nhanh này về Spark là cách sử dụng nó trong các chương trình độc lập. Ngoài việc chạy tương tác, Spark có thể được liên kết vào các ứng dụng độc lập trong Java, Scala hoặc Python. Sự khác biệt chính so với việc sử dụng nó trong shell là bạn cần khởi tạo SparkContext của riêng mình. Sau đó, API cũng giống nhau.

Quá trình liên kết đến Spark thay đổi tùy theo ngôn ngữ. Trong Java và Scala, bạn cung cấp cho ứng dụng của mình một phụ thuộc Maven trên tia lửa điện hiện vật. Tính đến thời điểm viết bài này, phiên bản Spark mới nhất là 1.2.0 và tọa độ Maven cho phiên bản đó là:

```
groupId = org.apache.spark  
artifactId = spark-core_2.10  
phiên bản = 1.2.0
```

Maven là một công cụ quản lý gói phổ biến cho các ngôn ngữ dựa trên Java cho phép bạn liên kết đến các thư viện trong kho lưu trữ công cộng. Bạn có thể sử dụng chính Maven để xây dựng dự án của mình hoặc sử dụng các công cụ khác có thể giao tiếp với kho lưu trữ Maven, bao gồm công cụ sbt của Scala hoặc Gradle. Các môi trường phát triển tích hợp phổ biến như Eclipse cũng cho phép bạn trực tiếp thêm phụ thuộc Maven vào một dự án.

Trong Python, bạn chỉ cần viết các ứng dụng dưới dạng tập lệnh Python, nhưng bạn phải chạy chúng bằng cách sử dụng `bin/spark-submit` tập lệnh được bao gồm trong Spark. `spark-submit` tập lệnh bao gồm các phụ thuộc Spark cho chúng ta trong Python. Tập lệnh này thiết lập môi trường để API Python của Spark hoạt động. Chỉ cần chạy tập lệnh của bạn với dòng được đưa ra trong [Ví dụ 2-6](#).

Ví dụ 2-6. Chạy một tập lệnh Python

```
bin/spark-submit my_script.py
```

(Lưu ý rằng bạn sẽ phải sử dụng dấu gạch chéo ngược thay vì dấu gạch chéo xuôi trên Windows.)

Khởi tạo SparkContext

Sau khi bạn đã liên kết một ứng dụng với Spark, bạn cần nhập các gói Spark vào chương trình của mình và tạo SparkContext. Bạn thực hiện bằng cách đầu tiên tạo một `Hội nghị SparkConf` đối tượng để cấu hình ứng dụng của bạn và sau đó xây dựng SparkContext cho nó. Ví dụ [2-7](#) bởi vì [2-9](#) thể hiện điều này bằng từng ngôn ngữ được hỗ trợ.

Ví dụ 2-7. Khởi tạo Spark trong Python

từ công viên pyspark nhập khẩu Hội nghị SparkConf, SparkContext

```
conf=Hội nghị SparkConf().thiết lậpMaster("địa phương").đặtAppName("Ứng  
dụng của tôi") sc=SparkContext(conf=conf)
```

Ví dụ 2-8. Khởi tạo Spark trong Scala

nhập khẩu `org.apache.spark.SparkConf` nhập
khẩu `org.apache.spark.SparkContext` nhập
khẩu `org.apache.spark.SparkContext._`

```
giá trị conf = mới Hội nghị SparkConf(). thiết lập Master("địa phương"). đặtAppName("Ứng dụng của  
tôi") giá trị sc = mới SparkContext(conf)
```

Ví dụ 2-9. Khởi tạo Spark trong Java

nhập khẩu `org.apache.spark.SparkConf`;
nhập khẩu `org.apache.spark.api.java.JavaSparkContext`;

```
hội nghị SparkConf = mới Hội nghị SparkConf(). thiết lập Master("địa phương"). đặtAppName("Ứng dụng  
của tôi"); JavaSparkContext sc = mới JavaSparkContext.Builder(conf);
```

Các ví dụ này cho thấy cách tối thiểu để khởi tạo `SparkContext`, trong đó bạn truyền hai tham số:

- **MỘT URL cụm**, cụ thể là địa phương trong các ví dụ này, điều này cho Spark biết cách kết nối với cụm. Địa phương là giá trị đặc biệt chạy Spark trên một luồng trên máy cục bộ, mà không cần kết nối với cụm.
- **MỘT tên ứng dụng**, cụ thể là ứng dụng của tôi trong các ví dụ này. Điều này sẽ xác định ứng dụng của bạn trên giao diện người dùng của trình quản lý cụm nếu bạn kết nối với cụm.

Có các tham số bổ sung để cấu hình cách ứng dụng của bạn thực thi hoặc thêm mã để chuyển đến cụm, nhưng chúng tôi sẽ đề cập đến những điều này trong các chương sau của cuốn sách.

Sau khi khởi tạo `SparkContext`, bạn có thể sử dụng tất cả các phương pháp chúng tôi đã trình bày trước đó để tạo RDD (ví dụ: từ tệp văn bản) và thao tác chúng.

Cuối cùng, để tắt Spark, bạn có thể gọi `đừng lại()` (phương pháp trên `SparkContext` của bạn) hoặc chỉ cần thoát khỏi ứng dụng (ví dụ: với `Hệ thống.thoát(0)` hoặc `sys.exit()`).

Tổng quan nhanh này đủ để bạn chạy ứng dụng Spark độc lập trên máy tính xách tay của mình. Để cấu hình nâng cao hơn, **Chương 7** sẽ đề cập đến cách kết nối ứng dụng của bạn với một cụm, bao gồm đóng gói ứng dụng của bạn để mã của nó được tự động chuyển đến các nút công nhân. Hiện tại, vui lòng tham khảo **Hướng dẫn bắt đầu nhanh** trong tài liệu chính thức của Spark.

Xây dựng các ứng dụng độc lập

Đây sẽ không phải là chương giới thiệu hoàn chỉnh của một cuốn sách về Dữ liệu lớn nếu chúng ta không có ví dụ về số lượng từ. Trên một máy duy nhất, việc triển khai số lượng từ rất đơn giản, nhưng trong các khuôn khổ phân tán, đây là một ví dụ phổ biến vì nó liên quan đến việc đọc và kết hợp dữ liệu từ nhiều nút công nhân. Chúng ta sẽ xem xét việc xây dựng và

đóng gói một ví dụ đếm từ đơn giản với cả sbt và Maven. Tất cả các ví dụ của chúng tôi có thể được xây dựng cùng nhau, nhưng để minh họa cho một bản dựng được tinh giản với các phụ thuộc tối thiểu, chúng tôi có một dự án nhỏ hơn riêng biệt bên dưới *ví dụ về học tập spark/ví dụ hoàn chỉnh nhỏ* thư mục, như bạn có thể thấy trong Ví dụ 2-10(Java) và 2-11 (Thang đo).

Ví dụ 2-10. Số lượng từ trong ứng dụng Java—đừng lo lắng về chi tiết ngay

```
// Tạo một ngữ cảnh Java Spark
hội nghị SparkConf=mớiHội nghị SparkConf().đặtAppName("Số từ");
JavaSparkContext sc=mớiJavaSparkBối cảnh(conf); // Tải dữ liệu đầu vào
của chúng ta.
JavaRDD<Sợi dây>đầu vào=sc.Tập văn bản(đầu
vàoFile); // Tách thành các từ.
JavaRDD<Sợi dây>từ=đầu vào.Bản đồ phẳng(
    mớiHàm FlatMap<Sợi dây,Sợi dây>() {
        công cộngCó thể lặp lại<Sợi dây>gọi (Chuỗi x) {
            trả lạiMảng.nhưDanh sách(x.tách ra(" "
            )); }});
// Biến đổi thành cặp và đếm.
JavaPairRDD<Sợi dây,Số nguyên>đếm=từ.mapToPair(
    mớiHàm cặp<Sợi dây,Sợi dây,Số nguyên>(){
        công cộngBộ 2<Sợi dây,Số nguyên>gọi (Chuỗi x){
            trả lại mớiBộ 2(x,1);
        }}).giảmTheoKhóa(mớiChức năng 2<Số nguyên,Số nguyên,Số nguyên>(){
        công cộngSố nguyêngọi (Số nguyên x,Số nguyên y){trở lạix+và;}});
// Lưu số lượng từ trở lại vào một tệp văn bản để đánh giá. đếm.lưu dưới
dạng tệp văn bản(đầu raFile);
```

Ví dụ 2-11. Số lượng từ trong ứng dụng Scala—đừng lo lắng về chi tiết ngay

```
// Tạo một ngữ cảnh Scala Spark.
giá trịconf=mớiHội nghị SparkConf().đặtAppName("Số từ") giá
trịsc=mớiSparkContext(conf) // Tải dữ liệu đầu vào của chúng
ta.
giá trịđầu vào=sc.Tập văn bản(đầu
vàoFile) // Chia nó thành các từ.
giá trịtừ=đầu vào.Bản đồ phẳng(đường kẻ=>đường kẻ.tách
ra(" ")) // Biến đổi thành cặp và đếm.
giá trịđếm=từ.bản đồ(từ=>(từ,1)).giảmTheoKhóa{trường hợp(x,và)=>x+và; // Lưu số
lượng từ trở lại vào một tệp văn bản để đánh giá. đếm.lưu dưới dạng tệp văn bản(
đầu raFile)
```

Chúng ta có thể xây dựng các ứng dụng này bằng cách sử dụng các tệp xây dựng rất đơn giản với cả sbt (Ví dụ 2-12) và Maven (Ví dụ 2-13). Chúng tôi đã đánh dấu sự phụ thuộc của Spark Core là cung cấp vì vậy, sau này, khi chúng ta sử dụng JAR lắp ráp, chúng ta không bao gồm tia lửa-löiJAR, vốn đã có trên classpath của các worker.

Ví dụ 2-12. tệp xây dựng sbt

```
tên:="ví dụ nhỏ về tia lửa học tập"

phiên bản:="0.0.1"

scalaPhiên bản:="2.10.4"

// thư viện bổ sung thư việnPhụ
thuộc+=Thứ tự(
  "org.apache.spark"%%"lõi tia lửa"%"1.2.0"%"cung cấp"
)
```

Ví dụ 2-13. Tệp xây dựng Maven

```
<dự án>
  <nhómId>com.oreilly.learningsparkexamples.mini</groupId>
  <mã hiện vật>ví dụ nhỏ về tia lửa học tập</artifactId>
  <modelPhiên bản>4.0.0</modelVersion> <tên>ví dụ</tên>

  <bao bì>cải lọ</bao bì>
  <phiên bản>0.0.1</phiên bản>
  <bản> <phụ thuộc>
    <phụ thuộc><!--Phụ thuộc Spark -->
      <nhómId>tổ chức.apache.spark</groupId> <mã
        hiện vật>spark-core_2.10</artifactId> <phiên
        bản>1.2.0</phiên bản> <phạm vi>cung cấp</
        phạm vi> </phụ thuộc>

    </phụ thuộc>
  <thuộc tính>
    <java.phiên bản>1.6</java.version>
  </properties>
  <xây dựng>
    <Quản lý plugin>
      <plugin>
        <phần bố trợ> <nhómId>org.apache.maven.plugins</groupId>
        <ID hiện vật>maven-compiler-plugin</artifactId>
        <phiên bản>3.1</phiên bản> <cấu hình>

        <nguồn>${java.phiên bản}</nguồn>
        <mục tiêu>${java.phiên bản}</target>
      </configuration> </plugin> </plugin> </
      plugins>
    </pluginQuản lý>
  </xây dựng>
</dự án>
```



Cát tia lửa-lôigói được đánh dấu lằcung cấptrong trường hợp chúng tôi đóng gói ứng dụng của mình vào một JAR lắp ráp. Điều này được đề cập chi tiết hơn trong **Chương 7**.

Sau khi chúng tôi đã xác định được bản dựng của mình, chúng tôi có thể dễ dàng đóng gói và chạy ứng dụng của mình bằng cách sử dụng `bin/spark-giữkịch bản.spark-giữtập lệnh` thiết lập một số biến môi trường được Spark sử dụng. Từ *mini-complete-ví dụ* thư mục chúng ta có thể xây dựng trong cả Scala (**Ví dụ 2-14**) và Java (**Ví dụ 2-15**).

Ví dụ 2-14. Xây dựng và chạy Scala

gói sạch sbt

```
$SPARK_TRANG CHỦ/bin/giữ tia lửa\
- - lớp com.oreilly.learningsparkexamples.mini.scala.WordCount\
./mục tiêu/...(như trên)\
./README.md ./số từ
```

Ví dụ 2-15. Xây dựng và chạy Maven

mvn sạch sẽ&&mvn biên dịch&&gói mvn

```
$SPARK_TRANG CHỦ/bin/giữ tia lửa\
- - lớp com.oreilly.learningsparkexamples.mini.java.WordCount\
./target/learning-spark-mini-example-0.0.1.jar\
./README.md ./số từ
```

Để biết thêm ví dụ chi tiết về việc liên kết các ứng dụng với Spark, hãy tham khảo **Hướng dẫn bắt đầu nhanh** trong tài liệu chính thức của Spark. **Chương 7** bao gồm việc đóng gói các ứng dụng Spark một cách chi tiết hơn.

Phần kết luận

Trong chương này, chúng tôi đã đề cập đến việc tải xuống Spark, chạy nó cục bộ trên máy tính xách tay của bạn và sử dụng nó theo cách tương tác hoặc từ một ứng dụng độc lập. Chúng tôi đã cung cấp một cái nhìn tổng quan nhanh về các khái niệm cốt lõi liên quan đến lập trình với Spark: một chương trình trình điều khiển tạo ra `SparkContext` và `RDD`, sau đó chạy các hoạt động song song trên chúng. Trong chương tiếp theo, chúng ta sẽ đi sâu hơn vào cách `RDD` hoạt động.