Module 3 Tuần 2
A Gentle Introduction to PySpark for Data Processing

Time-Series Team

Ngày 10 tháng 8 năm 2025

Buổi học thứ 5 (ngày 7/8/2025) được chia thành 4 phần chính nhằm giúp ta hiểu được Cách PySpark xử lý dữ liệu cho Big Data và Machine Learning:

- **Phần 1: Big Data là gì và vì sao lại cần nó**
- **Phần 2: Giải thích cơ chế hoạt động của Hadoop và Apache Spark**
- **Phần 3: Pyspark trong Python và SQL**
- **Phần 4: So sánh workflow của Sklearn và PySpark để dễ hiểu hơn**

Phần 1: Big Data là gì và vì sao lại cần nó

Vì sao lại cần Big Data trong thế giới hiện nay?

Khối lượng dữ liệu toàn cầu dự kiến sẽ đạt khoảng **180–163 zettabyte** vào năm 2025 [**idc2023**], để hình dung thì 1 chiếc điện thoại ngày nay có 128GB và 1 zettabytes bằng 1 nghìn tỷ GB, tương đương với 7.8 tỷ chiếc điện thoại. Vậy nên việc xử lý, thu thập và xử lý thông tin lớn như BigData hay không, không phải là 1 lựa chọn mà là **1 lĩnh vực bắt buộc phải hiểu** để không bị bỏ lại phía sau.

Để dễ hình dung, ví dụ khi ta viết một chương trình nhỏ để tìm kiếm tên ta trong danh sách 100 người, thì việc xử lý khá đơn giản — đọc dữ liệu, tìm tên, trả kết quả.

Nhưng nếu:

- Dữ liệu không phải 100 người, mà là **100 triệu bản ghi** (hoặc hơn).
- Dữ liệu nằm rải rác ở nhiều nơi: server ở Hà Nội, TP.HCM, Singapore...
- Dữ liệu có nhiều kiểu và một trong số đó còn là thông tin chưa được xử lý.
- Yêu cầu phải trả kết quả **trong vài giây**.

Lúc này, ta sẽ gặp hai lớp vấn đề:

Lớp 1 – Logic của chương trình: Đây là "bộ não" của ứng dụng:

- Xác định ứng dụng cần làm gì (tìm kiếm, lọc, tính toán...).
- Viết bằng ngôn ngữ như Python, Java, SQL,...
- Ví dụ: "Tìm tất cả khách hàng đã mua hàng trên 5 triệu đồng trong tháng 7".

Lớp 2 – Thực thi trên dữ liệu khổng lồ: Đây là phần khó khi dữ liệu rất lớn:

- **Chia nhỏ dữ liệu** để xử lý song song.
- **Phân việc** cho nhiều máy chủ khác nhau.
- **Tối ưu hạ tầng** để mọi máy hoạt động trơn tru.
- **Tổng hợp kết quả** từ nhiều máy và trả về cho người dùng.
- **Xử lý lỗi**: nếu một máy hỏng, hệ thống vẫn tiếp tục chạy.

Như khi ta tra cứu Google, không phải một máy tính duy nhất tìm kiếm toàn bộ Internet. Hàng trăm máy sẽ chạy song song, mỗi máy xử lý một phần dữ liệu, rồi kết quả được gom lại cực nhanh.

2.1 Big Data là gì ?

Dữ liệu trở thành Big Data khi đạt đủ 5 tiêu chí **5V**):

1. **Volume** – Khối lượng dữ liệu khổng lồ.
   Ví dụ: Facebook lưu trữ hàng tỷ bức ảnh.
2. **Velocity** – Tốc độ dữ liệu được tạo ra và xử lý.
   Ví dụ: Dữ liệu livestream, giao dịch ngân hàng theo thời gian thực.
3. **Variety** – Sự đa dạng về định dạng và nguồn dữ liệu, dữ liệu có cấu trúc hoặc không hoặc bán cấu trúc.
   Ví dụ: Văn bản, video, ảnh, log server.
4. **Veracity** –Độ tin cậy và tính chính xác của dữ liệu.
   Ví dụ: Bộ dữ liệu y tế có thể chứa thông tin nhập sai hoặc trùng lặp.
5. **Value** – Giá trị mà dữ liệu đó mang lại.
   Ví dụ:
   - Cải thiện trải nghiệm khách hàng qua phân tích hành vi mua hàng để gợi ý sản phẩm phù hợp cho từng khách hàng.
   - Giảm chi phí qua phân tích những vấn đè, yếu tố lớn nhất đang kéo doanh thu xuống mà không mang lại lợi ích gì.
   - Giải quyết vấn đề tốt hơn qua hình dung vấn đề trong dữ liệu.
   - Gia tăng lợi nhuận nhờ vào phân tích mảng khách hàng cần quan tâm.

2.2 Tại sao cần các giải pháp như Apache Spark?

Các nền tảng như **Apache Spark** cho phép lập trình viên:

- Tập trung vào **Lớp 1** (logic xử lý) thay vì tự viết toàn bộ **Lớp 2**.
- Tự động chia dữ liệu và phân công cho nhiều máy xử lý.
- Quản lý tài nguyên và tổng hợp kết quả.
- Hỗ trợ cả dữ liệu **batch** và **streaming** (thời gian thực).

Nói cách khác: ta chỉ cần "ra mệnh lệnh" và Spark sẽ huy động "đội quân máy tính" để làm phần còn lại.

Phần 2: Giải thích cơ chế hoạt động của Hadoop và Apache Spark

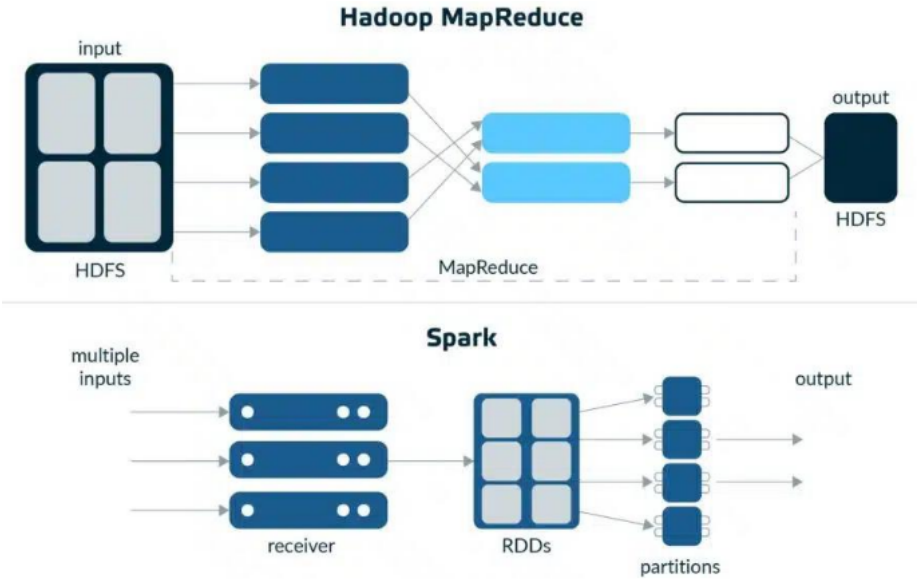
3.1 Mục Đích của Hadoop và Apache Spark

Từ ví dụ ở phần trước, khi khối lượng dữ liệu lên đến hàng trăm triệu bản ghi, phân tán ở nhiều máy chủ và yêu cầu phản hồi trong vài giây, ta cần đến các *framework* xử lý dữ liệu phân tán.

Parameter	Hadoop	Spark
Intent	Data Processing	Data Analytics
Work Process	Analyses batches of data present in huge volumes	Analyses and processes real-time data
Processing Style	Batch	Real-time
Latency	High	Low
Cost	Less costly due to the MapReduce Model	Is more expensive (in-memory solution)
Ease of Use	Complex to use	Easier to use
Written	Java	Scala

Hình 1: Hadoop and Apache comparison

- **Hadoop** phù hợp với *batch processing* khối lượng lớn, tận dụng mô hình **HDFS + MapReduce** để **lưu trữ và xử lý dữ liệu theo Batch**.
- **Apache Spark** kế thừa và mở rộng ý tưởng từ Hadoop, sử dụng *in-memory computing* giúp xử lý nhanh hơn, hỗ trợ cả *batch* và *streaming* dữ liệu, thuận tiện hơn cho **phân tích dữ liệu thời gian thực.**



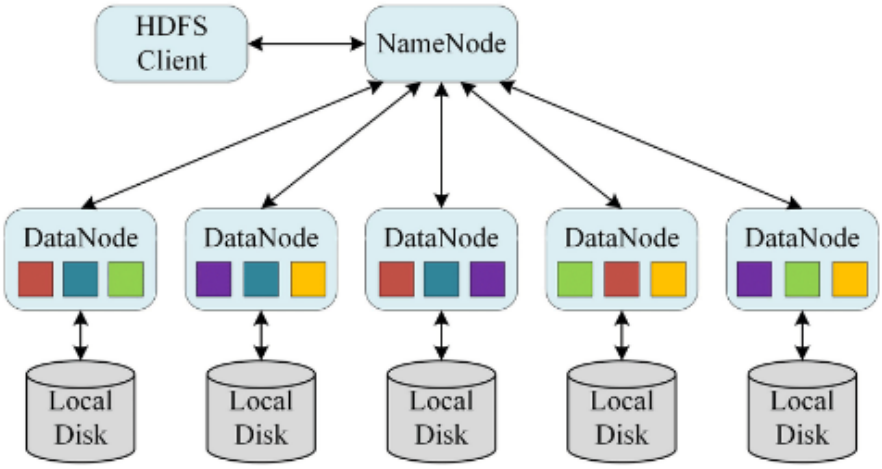
Hình 2: Minh họa workflow của Hadoop và Spark

Tóm lại, để xử lý dữ liệu lớn ta cần 1 hệ thống chuyển lưu trữ rồi xử lý dữ liệu theo Batch và 1 hệ thống cho phép phân tích dữ liệu theo thời gian thực sử dụng RAM. Giống như Ổ Cứng và RAM trong máy tính.

3.2 HDFS (Hadoop Distributed File System)

HDFS là hệ thống lưu trữ phân tán được thiết kế để xử lý dữ liệu lớn trên nhiều máy chủ. Toàn bộ work-flow của HDFS có thể tóm tắt như sau: khi **HDFS Client gửi yêu cầu đọc hoặc ghi dữ liệu**, HDFS Client

sẽ giao tiếp với NameNode để lấy metadata (cơ bản nó là 1 json string chứa chứa nhiều cặp keys và values) và thông tin vị trí các block dữ liệu xanh đỏ tím vàng như minh họa. Sau đó, **Client giao tiếp trực tiếp với các DataNode** để lấy hoặc ghi dữ liệu. Lưu ý là các block được nhân bản (replication) trên nhiều DataNode để đảm bảo dữ liệu vẫn khả dụng ngay cả khi một máy bị hỏng.



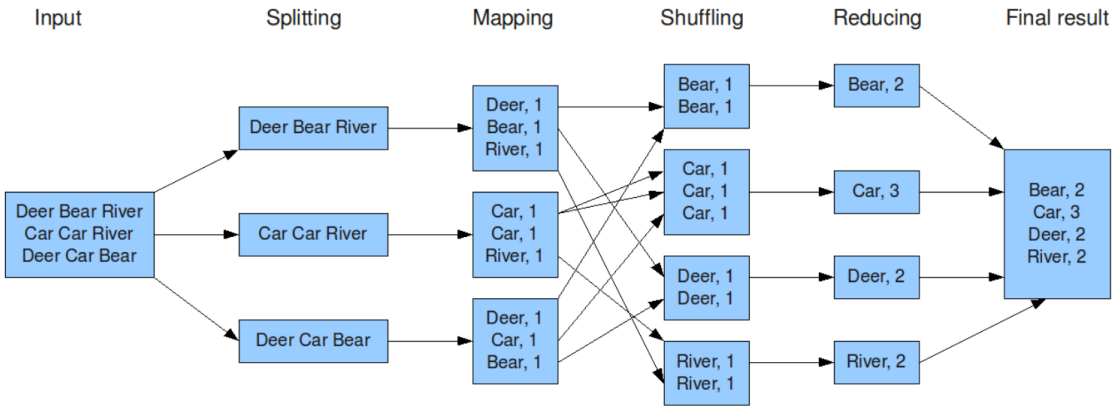
Hình 3: Cấu trúc HDFS

Các thành phần chính:

- **NameNode**: "Bộ não" của HDFS, quản lý metadata, vị trí các block, và điều phối việc đọc/ghi.
- **DataNode**: Lưu trữ dữ liệu thực tế dưới dạng block, mỗi DataNode chạy trên một máy chủ riêng.
- **HDFS Client**: Gửi yêu cầu đến NameNode và truy cập dữ liệu từ DataNode.
- **Replication**: Mỗi block được nhân bản trên nhiều DataNode để tăng khả năng chịu lỗi và độ sẵn sàng.

3.3 MapReduce

MapReduce là mô hình xử lý dữ liệu lớn theo cơ chế chia nhỏ và tổng hợp. Workflow tổng quát: dữ liệu đầu vào được chia nhỏ (**Splitting**) thành nhiều phần và xử lý song song bởi các hàm `Map`, tạo ra các cặp `(key, value)`. Sau đó, hệ thống thực hiện **Shuffling** để nhóm các giá trị cùng key lại với nhau, và cuối cùng `Reduce` tổng hợp chúng để tạo ra kết quả cuối cùng.



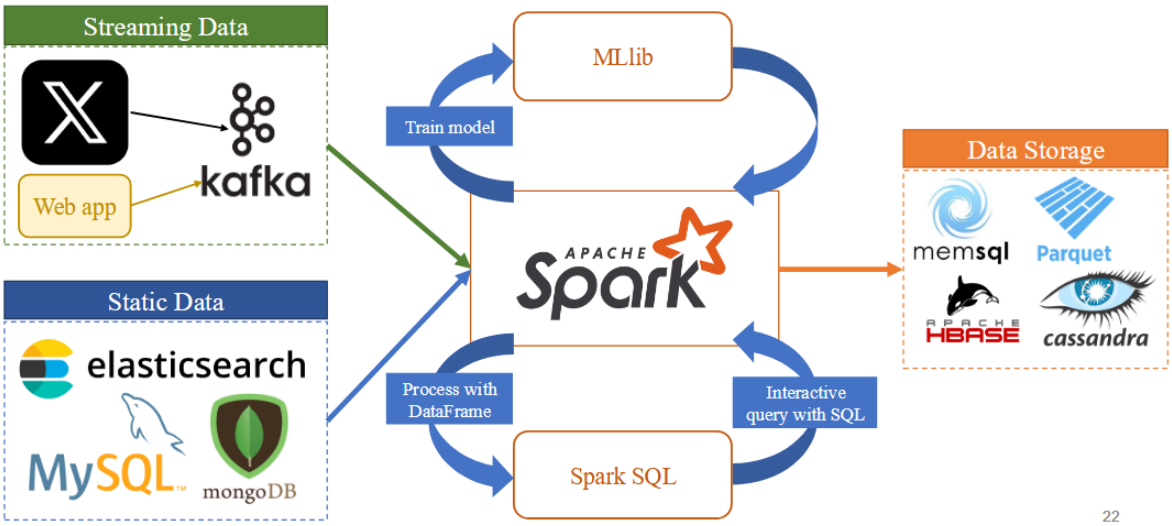
Hình 4: Quy trình MapReduce

Ví dụ các bước lưu trữ thông tin qua ví dụ đếm số chữ, giống ‘token_dict‘’ trong NLP:

1. **Splitting**: Chia dữ liệu thành các chunk nhỏ để xử lý song song.
2. **Mapping**: Hàm `Map` xử lý từng chunk và phát ra các cặp `(key, value)`.
3. **Shuffling**: Gom nhóm các giá trị có cùng `key`.
4. **Reducing**: Hàm `Reduce` xử lý từng nhóm `key` để tạo kết quả.
5. **Final Result**: Tổng hợp và xuất ra kết quả cuối cùng.

3.4 Tổng quan Pipeline hoạt động của Apache Spark

Để hiểu rõ hơn Apache Spark có hoạt động như thế nào, mình sẽ đi qua 1 workflow xử lý dữ liệu cho Học Máy, Flow bắt đầu từ việc thu thập dữ liệu từ nhiều nguồn khác nhau (streaming: dữ liệu được cập nhật liên tục và static: dữ liệu đã lưu), sau đó xử lý bằng các API như DataFrame hoặc Spark SQL, thực hiện các tác vụ phân tích nâng cao (MLlib), và cuối cùng lưu trữ kết quả (i.e. dữ liệu đã xử lý) vào các hệ thống lưu trữ dữ liệu.



Hình 5: Pipeline xử lý dữ liệu với PySpark

Các bước chính trong pipeline:

1. **Streaming Data**: Thu thập dữ liệu thời gian thực từ các nguồn như Kafka (ví dụ: dữ liệu mạng xã hội, log từ web app).
2. **Static Data**: Đọc dữ liệu tĩnh từ các hệ quản trị cơ sở dữ liệu hoặc công cụ tìm kiếm như Elasticsearch, MySQL, MongoDB.
3. **Process with DataFrame**: Sử dụng API DataFrame của Spark để xử lý dữ liệu một cách linh hoạt và tối ưu, tận dụng khả năng *in-memory computing*.
4. **Spark SQL**: Thực hiện các truy vấn tương tác với dữ liệu thông qua cú pháp SQL.
5. **MLlib**: Áp dụng các thuật toán học máy như phân loại, hồi quy, clustering để phân tích và trích xuất thông tin.
6. **Data Storage**: Lưu kết quả phân tích vào các hệ thống lưu trữ phân tán như MemSQL, Apache HBase, Cassandra hoặc định dạng file tối ưu như Parquet.

3.5 Pyspark Ecosystem

Để vận hành và xử ý dữ liệu trơn tru cho các mục đích khác nhau, các hệ sinh thái được xây dựng lên xung quanh Apache Spark:

- Storage and Infrastructure đóng vai trò là ”nền móng”, nơi lưu trữ và quản lý dữ liệu khổng lồ (ví dụ: Hadoop, S3, Parquet).
- Data Science and Machine Learning là ”công cụ”để các Data Scientist xây dựng các mô hình phức tạp, phân tích dữ liệu chuyên sâu và trích xuất thông tin giá trị từ dữ liệu thô/raw.
- SQL Analytics and BI là ”giao diện”, nơi dữ liệu đã được xử lý và phân tích được trình bày dưới dạng trực quan, giúp khách hàng dễ dàng hiểu và đưa ra quyết định.

Lưu ý: mô tả thư viện dưới đây chỉ mang tính chất tham khảo, để hiểu chỉ cần đọc mô tả của từng phần.
Data science and Machine learning: sử dụng để mở rộng khả năng xử lý dữ liệu lớn bằng cách tích hợp các mô hình và thuật toán học máy.

- scikit-learn: Một bộ công cụ dành cho lập trình viên Python để xây dựng các mô hình học máy một cách đơn giản, ví dụ như dự đoán xu hướng hoặc phân loại dữ liệu.
- pandas: Giống như một bảng tính Excel siêu mạnh trong Python, giúp ta dễ dàng làm sạch, xử lý và phân tích dữ liệu.
- TensorFlow: Một nền tảng mạnh mẽ giúp ta tạo ra các mô hình học máy phức tạp, đặc biệt là các mô hình sử dụng mạng lưới nơ-ron để làm những việc như nhận diện hình ảnh hoặc dịch ngôn ngữ.
- PyTorch: Một công cụ khác, cũng rất phổ biến để xây dựng các mô hình học máy, được nhiều nhà nghiên cứu ưa chuộng vì tính linh hoạt và dễ sử dụng.
- mlflow: Một công cụ giúp ta quản lý toàn bộ quá trình phát triển các mô hình học máy, từ việc thử nghiệm các thuật toán khác nhau cho đến khi triển khai chúng.
- R: Một ngôn ngữ lập trình chuyên biệt dành cho các nhà thống kê và nhà khoa học dữ liệu để phân tích dữ liệu và tạo ra các biểu đồ đẹp mắt.
- NumPy: Nền tảng cơ bản cho hầu hết các thư viện khoa học dữ liệu trong Python, giúp xử lý các phép toán với số và ma trận một cách cực kỳ nhanh chóng.

SQL analytics and BI: dùng để trực quan hóa và phân tích dữ liệu đã được xử lý hoặc lưu trữ, giúp tạo ra báo cáo và bảng điều khiển hữu ích.

- Apache Superset: Một công cụ mã nguồn mở giúp ta tạo ra các bảng điều khiển (dashboard) và biểu đồ tương tác để khám phá dữ liệu của mình mà không cần phải viết nhiều code.
- Power BI: Một dịch vụ của Microsoft giúp ta biến dữ liệu thô thành những báo cáo và biểu đồ dễ hiểu, giúp ta đưa ra quyết định kinh doanh tốt hơn.
- Looker: Một nền tảng phân tích dữ liệu giúp các công ty dễ dàng truy cập và hiểu dữ liệu của họ, đặc biệt là khi dữ liệu phức tạp.
- re:dash: Một công cụ mã nguồn mở giúp ta tạo các bảng điều khiển trực quan và chia sẻ kết quả truy vấn dữ liệu với đồng nghiệp.

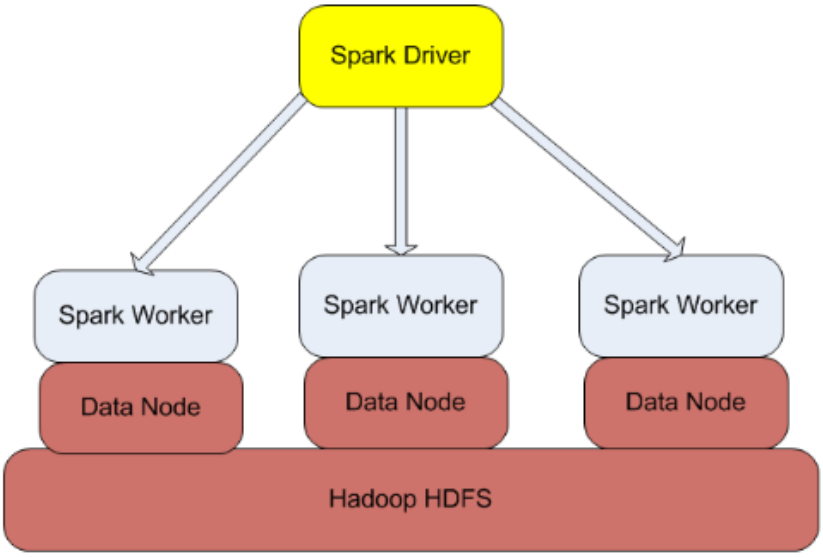
- tableau: Một trong những công cụ hàng đầu trong việc trực quan hóa dữ liệu, giúp ta tạo ra những biểu đồ và đồ thị tương tác một cách nhanh chóng.
- dbt: Một công cụ dành cho các kỹ sư dữ liệu để biến đổi dữ liệu thô trong kho dữ liệu thành các bảng đã được làm sạch và chuẩn bị sẵn sàng cho việc phân tích.

Storage and Infrastructure: đóng vai trò quan trọng trong việc lưu trữ, quản lý và xử lý dữ liệu lớn, tạo nền tảng vững chắc cho các tác vụ phân tích và học máy.

- elasticsearch: Giống như một công cụ tìm kiếm Google cho dữ liệu của ta, giúp ta tìm kiếm thông tin một cách nhanh chóng và mạnh mẽ.
- mongoDB: Một loại cơ sở dữ liệu linh hoạt, không cần tuân theo cấu trúc bảng cố định như Excel, rất phù hợp để lưu trữ dữ liệu đa dạng như thông tin người dùng trên mạng xã hội.
- Apache Kafka: Một ”ống dẫn”dữ liệu tốc độ cao, dùng để chuyển tiếp một lượng lớn dữ liệu theo thời gian thực từ nhiều nguồn khác nhau đến nơi cần xử lý.
- Apache Airflow: Một công cụ để lên lịch và quản lý các công việc phức tạp, giúp ta đảm bảo các quy trình xử lý dữ liệu luôn chạy đúng thứ tự và đúng thời điểm.
- Parquet: Một cách để lưu trữ dữ liệu hiệu quả, đặc biệt là khi ta chỉ cần truy cập một vài cột dữ liệu trong một tập tin lớn.
- Microsoft SQL Server: Một hệ thống quản lý cơ sở dữ liệu truyền thống, rất mạnh mẽ và ổn định, được sử dụng rộng rãi trong các doanh nghiệp.
- Delta Lake: Một ”lớp phủ”giúp cải thiện độ tin cậy và hiệu suất của các kho dữ liệu lớn, giúp dữ liệu luôn nhất quán và dễ quản lý hơn.
- kubernetes: Một công cụ giúp ta quản lý hàng trăm, thậm chí hàng ngàn, ứng dụng nhỏ (gọi là container) một cách tự động, đảm bảo chúng luôn chạy ổn định.
- cassandra: Một loại cơ sở dữ liệu phân tán, được thiết kế để xử lý lượng dữ liệu khổng lồ trên nhiều máy chủ mà không bao giờ bị lỗi.
- Apache ORC: Một định dạng lưu trữ dữ liệu khác, tương tự Parquet, cũng giúp tối ưu hóa việc truy cập và xử lý dữ liệu.

3.6 Kiến trúc của Apache Spark

Phần này mô tả workflow tổng quát của Spark theo kiến trúc top-down: bắt đầu từ **Spark Driver** gửi công việc xuống các **Spark Worker**, mỗi worker gắn với một **Data Node** lưu dữ liệu trên **Hadoop HDFS**. Tại mỗi worker, dữ liệu được xử lý bởi **Executor** thông qua các **Task** thực thi trên các **Partition** dữ liệu.



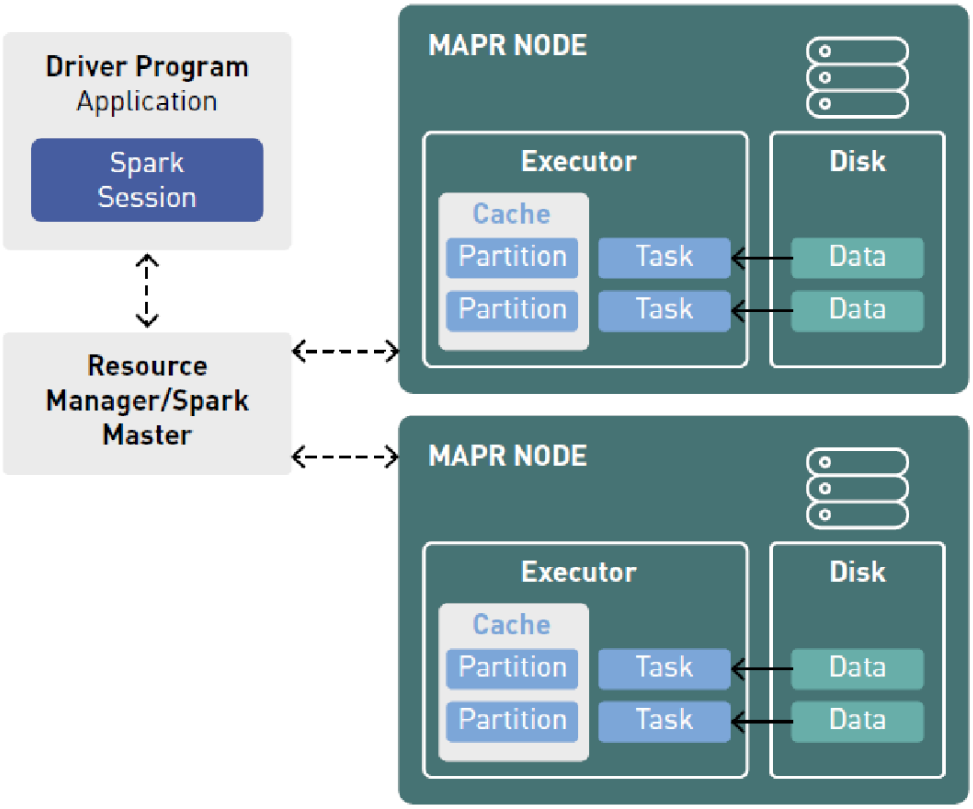
Hình 6: PySpark top-down architecture

Workflow tổng quát.

1. **Spark Driver** khởi tạo ứng dụng, xây dựng kế hoạch xử lý (job) và gửi yêu cầu tới các **Spark Worker**.
2. Mỗi **Spark Worker** liên kết với một **Data Node** để truy xuất dữ liệu từ **HDFS**.
3. Worker khởi chạy **Executor**, chia dữ liệu thành các **Partition** và gán **Task** cho từng partition.
4. Các Task chạy song song trên nhiều node, sau đó trả kết quả về Spark Driver hoặc ghi ra storage.

Thành phần chính.

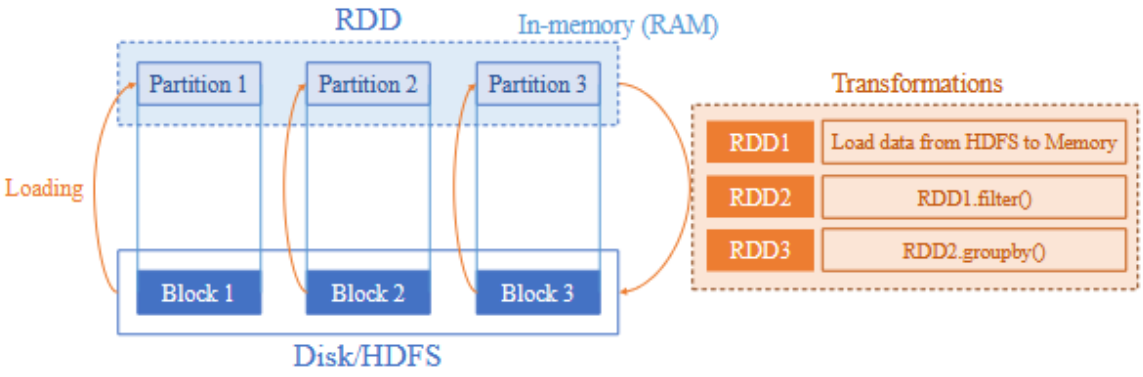
- **Spark Driver**: Điều phối toàn bộ luồng xử lý.
- **Spark Worker**: Node tính toán, nhận lệnh từ Driver.
- **Executor**: Tiến trình trên Worker thực hiện các Task.
- **Data Node**: Nơi lưu trữ dữ liệu vật lý, thường là các block trong HDFS.
- **HDFS**: Hệ thống lưu trữ phân tán cung cấp dữ liệu cho Spark.



Hình 7: Ví dụ chi tiết kiến trúc Spark: Driver, Resource Manager, Executors và Data Nodes

3.7 Apache Spark: RDD (Resilient Distributed Dataset)

RDD là cấu trúc dữ liệu cốt lõi của Spark, được phân tán (**Distributed**) qua nhiều node, có khả năng chịu lỗi (**Resilient**) và chứa dữ liệu dưới dạng các **Partition**. Phần này mô tả workflow RDD từ khi dữ liệu nằm trên **Disk/HDFS** đến khi được xử lý trong **RAM** thông qua các **Transformations**.



Hình 8: PySpark RDD: Partition từ Disk/HDFS và xử lý In-memory

Workflow tổng quát.

1. **Loading**: Dữ liệu được đọc từ **Disk/HDFS** dưới dạng các **Block**.
2. Mỗi Block trong ổ cứng ánh xạ thành một **Partition** trong RDD và được load vào **In-memory (RAM)**.

3. Các **Transformations** (ví dụ: `filter()`, `groupBy()`) được áp dụng lên partition nhưng chỉ lưu dưới dạng lineage (lazy) - nghĩa là Spark lưu lại lịch sử thực thi chứ chưa thực thi ngay.

   Dựa trên **lineage**, Spark xây dựng một **DAG** (Directed Acyclic Graph) thể hiện mối quan hệ phụ thuộc giữa các bước xử lý. DAG là *kế hoạch thực thi* được Spark tối ưu hóa trước khi chạy, giúp:

   - Xác định thứ tự các phép tính cần thực hiện.
   - Chia công việc thành các **stage** và **task** để xử lý song song.
   - Tái tính toán dữ liệu nếu một partition bị mất, bằng cách chạy lại các bước cần thiết trong DAG.

   → Nói chung, lineage được tạo ra để lưu lịch sử thực thi và DAG được tạo ra để nắm bắt mqh giữa các bước thực thi dể khi chạy lỗi sẽ không phải chạy lại.

4. Khi gặp một **Action** (ví dụ: `count()`, `collect()`), Spark thực thi pipeline: chia thành **Tasks** chạy song song trên các partition.

5. Kết quả được trả về hoặc ghi lại xuống storage; các partition có thể được **Cache** để tái sử dụng.

Thành phần chính.

- **Block**: Đơn vị lưu trữ của HDFS.
- **Partition**: Đơn vị tính toán song song của Spark, ánh xạ từ Block.
- **In-memory (RAM)**: Nơi Spark giữ partition đã load để giảm I/O.
- **Transformations**: Các phép biến đổi tạo RDD mới một cách lazy.
- **Actions**: Kích hoạt thực thi và trả kết quả.

Phần 3: PySpark trong Python và SQL

Phần này trình bày các **component** chính của PySpark và cách chúng tích hợp Spark SQL và Spark MLlib trong xử lý dữ liệu phân tán.

4.1 Spark SQL

Spark SQL tích hợp truy vấn SQL với sức mạnh xử lý phân tán của Spark:

- Hỗ trợ nhiều định dạng dữ liệu: Parquet, JSON, CSV, Elasticsearch, JDBC.
- DataFrame cung cấp API thống nhất để làm việc với dữ liệu có cấu trúc và bán cấu trúc.
- Workflow: Load dữ liệu → xử lý truy vấn với SQL → xuất DataFrame kết quả.

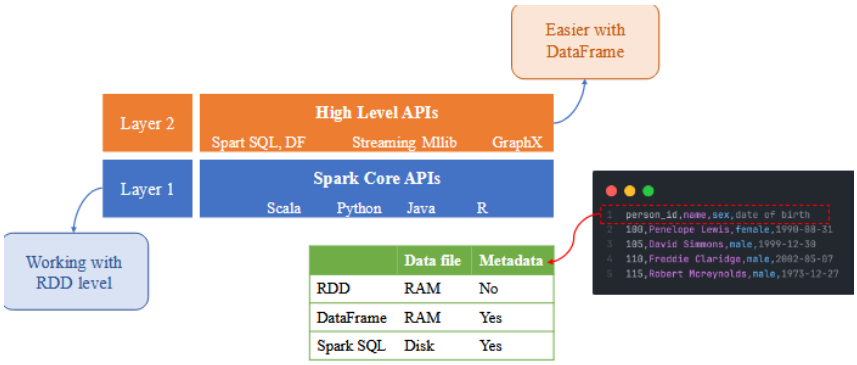
4.2 Spark MLlib

Spark MLlib cung cấp thư viện Machine Learning phân tán:

- Thuật toán: classification, regression, clustering, collaborative filtering.
- Feature engineering: feature extraction, transformation, dimensionality reduction.
- ML Pipeline: ingest dữ liệu → tiền xử lý → huấn luyện mô hình → kiểm thử → deploy.

4.3 PySpark Layers

PySpark được thiết kế theo hai tầng chính:



Hình 9: PySpark Layer

- **Layer 1 – Spark Core APIs:** Cung cấp API nền tảng cho nhiều ngôn ngữ (Scala, Python, Java, R). Ở mức này, lập trình viên làm việc trực tiếp với **RDD** (Resilient Distributed Dataset) - viết hàm, tập trung vào các thao tác low-level như `map()`, `filter()`, `reduceByKey()`, hoặc 1 Custom RDD.
- **Layer 2 – High Level APIs:** Bao gồm các API nâng cao như **Spark SQL / DataFrame**, **Spark Streaming**, **MLlib**, **GraphX** giúp đơn giản hóa việc xử lý dữ liệu bằng cách cung cấp cú pháp dễ đọc, hỗ trợ metadata, và tích hợp nhiều chức năng xử lý dữ liệu.

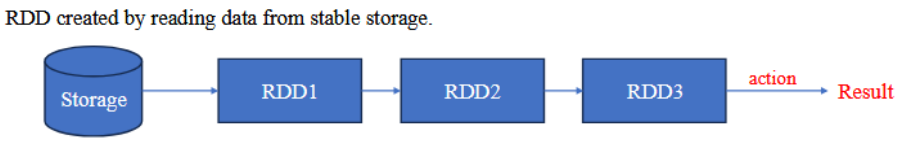
So sánh:

- **RDD:** Lưu dữ liệu trong RAM, không có metadata.
- **DataFrame:** Lưu dữ liệu trong RAM, có metadata (schema), tối ưu cho truy vấn và thao tác dạng bảng.
- **Spark SQL:** Lưu trên disk, có metadata, thích hợp cho phân tích dữ liệu lớn với truy vấn SQL.

4.4 PySpark RDD

RDD là cấu trúc dữ liệu nền tảng của Spark, biểu diễn tập dữ liệu phân tán, có khả năng chịu lỗi, và bất biến (immutable). Mỗi **RDD** có thể được xem như một *function* áp dụng lên dữ liệu đầu vào.

1. **Transformations:** (ví dụ: `map()`, `filter()`, `flatMap()`, `union()`) — tạo ra một RDD mới từ RDD gốc, nhưng không thực thi ngay (lazy evaluation). Spark lưu lại lịch sử thực thi (lineage) để tối ưu và phục hồi dữ liệu khi cần.
2. **Actions:** (ví dụ: `count()`, `collect()`, `saveAsTextFile()`) — kích hoạt việc thực thi pipeline các transformations và trả về kết quả.



Hình 10: PySpark RDD Workflow in Code

Ví dụ code minh họa:

```
# Tạo SparkContext
from pyspark import SparkContext
sc = SparkContext("local", "RDD Example")

# Tạo RDD từ list Python
data = [1, 2, 3, 4, 5]
rdd = sc.parallelize(data)

# Transformation: nhân đôi mỗi phần tử
rdd2 = rdd.map(lambda x: x * 2)

# Transformation: lọc số > 5
rdd3 = rdd2.filter(lambda x: x > 5)

# Action: Thu kết quả về driver
result = rdd3.collect()
print(result)  # [6, 8, 10]
```

Trong ví dụ trên:

- `map()` và `filter()` là transformations → chỉ tạo lineage, chưa thực thi.
- `collect()` là action → kích hoạt Spark chạy pipeline và trả kết quả.

4.5 Pyspark Connect Code

Phiên bản pyspark nên cài để tránh xảy ra Lỗi là v3.5.x, không nên cài v4.0.0: Mình có thể chạy PySpark trực tiếp qua Jypyter Notebook

```
!pip íntall pyspark==3.5.1

from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
spark
```

Tham khảo mã nguồn ví dụ PySpark:
Thực hành xử lý dữ liệu qua code PySpark để hiểu rõ hơn (chăm nghe ko bằng 1 thấy mà):
https://drive.google.com/drive/folders/1C8LUx7UaSkM3k4r4PsZFBJOt2bLGHCnj

4.6 1 Chút về Tối ưu Pyspark với DAG

DAG (Directed Acyclic Graph) là biểu đồ có hướng và không có chu trình, biểu diễn luồng xử lý dữ liệu trong Spark. Khi lập trình viên áp dụng các **Transformations** (ví dụ: `select()`, `map()`, `filter()`, `udf()`), Spark sẽ không thực thi ngay, mà xây dựng một **Execution Plan** dưới dạng DAG — thể hiện thứ tự và mối quan hệ giữa các bước xử lý.

Lazy Evaluation: Spark áp dụng cơ chế **Lazy Evaluation** — chỉ thực thi khi gặp một **Action** (ví dụ: `count()`, `collect()`). Điều này giúp Spark:

- Tối ưu kế hoạch thực thi (DAG Optimization) để giảm số lần đọc/ghi dữ liệu.
- Tránh việc tính toán lặp lại.

Cache vs Persist

❖ **Cache**

```
from pyspark import StorageLevel

df.persist(StorageLevel.MEMORY_AND_DISK)
```

StorageLevel	Speed	Location	Read
MEMORY_ONLY	fastest	No	RAM
MEMORY_AND_DISK	fast	RAM when low	RAM or Disk
DISK_ONLY	low	Disk	Disk

Hình 11: PySpark Storage Optimization with Stoage Level

- **Cache:** Lưu dữ liệu vào RAM (`MEMORY_ONLY`) — phù hợp cho dữ liệu vừa khít bộ nhớ.
- **Persist:** Cho phép chọn **Storage Level** (ví dụ: `MEMORY_AND_DISK`, `DISK_ONLY`) — dùng khi dữ liệu quá lớn để lưu toàn bộ trong RAM.

Thực tế: `Cache()` chính là `Persist()` với mặc định `MEMORY_ONLY`.

Checkpoint được sử dụng khi:

- DAG quá dài, gây rủi ro mất dữ liệu lineage khi node bị lỗi.
- Cần lưu trạng thái trung gian đáng tin cậy xuống HDFS hoặc storage ổn định.

Khác với Cache/Persist (lưu tạm thời trong bộ nhớ hoặc ổ đĩa local), Checkpoint sẽ ghi dữ liệu xuống hệ thống lưu trữ bền vững (thường là HDFS).

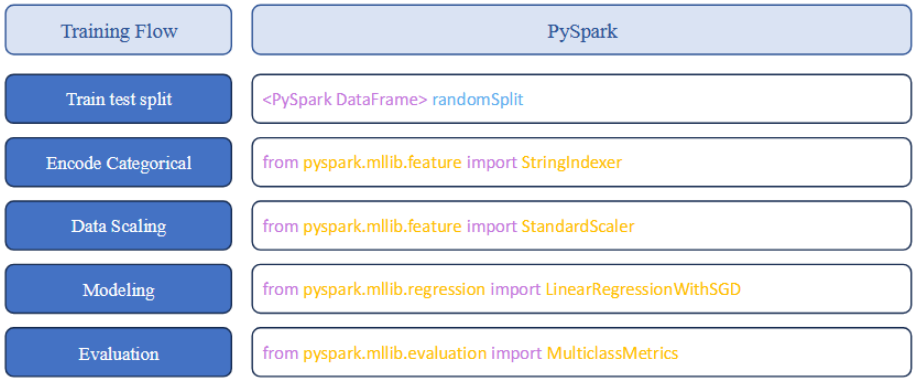
Phần 4: So sánh workflow của Sklearn và PySpark để dễ hiểu hơn

Sklearn

Training Flow	Sklearn
Train test split	from sklearn.model_selection import train_test_split
Encode Categorical	from sklearn.preprocessing import LabelEncoder
Data Scaling	from sklearn.preprocessing import StandardScaler
Modeling	from sklearn.linear_model import LinearRegression
Evaluation	from sklearn.metrics import accuracy_score

Hình 12: Sklearn Training workflow

PySpark



Hình 13: Pyspark Training workflow