1. **Phần Giới Thiệu.**

**1.1. Lý do chọn đề tài.**

Phân tích log máy chủ web là một công việc quan trọng để hiểu hành vi người dùng và đánh giá hiệu suất của các ứng dụng web. Các tệp log này chứa nhiều thông tin giá trị về hành vi người dùng, tình trạng hệ thống và các chỉ số hiệu suất tổng thể, vì vậy chúng đóng vai trò không thể thiếu trong việc tối ưu hóa dịch vụ web. Với sự gia tăng theo cấp số nhân của lưu lượng truy cập web, nhu cầu về một hệ thống xử lý dữ liệu mạnh mẽ, có khả năng quản lý và phân tích hiệu quả lượng lớn dữ liệu log trở nên vô cùng cấp thiết.

Để đáp ứng nhu cầu đó, dự án của chúng tôi hướng tới việc xây dựng một quy trình xử lý log máy chủ web tiên tiến, tận dụng khả năng của Hadoop và Spark. Hệ thống này được thiết kế để thu thập, lưu trữ và phân tích dữ liệu log một cách hiệu quả, từ đó cung cấp những thông tin chi tiết có thể hành động, góp phần cải thiện hiệu suất máy chủ web và trải nghiệm người dùng.

Hadoop, với hệ thống lưu trữ phân tán HDFS, cung cấp giải pháp lưu trữ có khả năng mở rộng và chống lỗi, rất phù hợp cho việc xử lý các tập dữ liệu lớn. Bổ sung cho Hadoop, Apache Spark mang lại khả năng xử lý dữ liệu nhanh chóng, mạnh mẽ và trong bộ nhớ, rất cần thiết cho việc phân tích thời gian thực.

Quy trình nhập dữ liệu trong hệ thống của chúng tôi sử dụng Apache Flume để chuyển dữ liệu log vào HDFS một cách liền mạch. Sau khi được nhập, dữ liệu sẽ trải qua một chuỗi các bước xử lý bằng PySpark như phân tích cú pháp, làm sạch và cấu trúc lại các mục log để sẵn sàng cho việc phân tích. Dữ liệu đã được cấu trúc sau đó sẽ được phân tích bằng Apache Hive, cho phép chúng tôi thực hiện các truy vấn tương tự SQL để rút ra các thông tin hữu ích như phân bố mã phản hồi HTTP, số lượng yêu cầu từ khách hàng, và thời gian phản hồi trung bình.

Việc trực quan hóa kết quả của các truy vấn này giúp trình bày phát hiện một cách dễ hiểu, từ đó hỗ trợ quá trình ra quyết định hiệu quả hơn. Dự án này tận dụng sức mạnh của các công nghệ dữ liệu lớn để xây dựng một hệ thống xử lý log máy chủ web hiệu quả và có khả năng mở rộng, góp phần nâng cao hiệu suất và trải nghiệm người dùng.

**1.2. Mục tiêu của đồ án.**

Động lực chính của đồ án này là nhu cầu ngày càng tăng trong việc quản lý và phân tích hiệu quả khối lượng lớn dữ liệu log từ máy chủ web. Việc này đặc biệt quan trọng để tối ưu hóa hiệu suất máy chủ và nâng cao hiệu quả vận hành. Khi các tổ chức và dịch vụ web ngày càng tạo ra lượng lớn dữ liệu log mỗi ngày, nhu cầu về các giải pháp mạnh mẽ có thể xử lý và phân tích dữ liệu này để rút ra những thông tin có thể hành động và hỗ trợ quá trình ra quyết định ngày càng trở nên cấp thiết.

Mục tiêu của dự án là tận dụng một tập hợp các công nghệ dữ liệu lớn — bao gồm Hadoop, Flume, Spark và Hive — để phát triển một giải pháp toàn diện cho việc xử lý và phân tích log máy chủ web. Qua đó, chúng tôi mong muốn chứng minh được tính ứng dụng thực tiễn của các công nghệ này trong các bài toán xử lý dữ liệu quy mô lớn. Những thông tin thu được từ quá trình phân tích sẽ cung cấp cái nhìn giá trị về mức độ sử dụng và hiệu suất máy chủ, từ đó hỗ trợ xây dựng các chiến lược quản lý hiệu quả hơn.

Dự án không chỉ minh họa tính ứng dụng thực tế của các công nghệ tiên tiến này mà còn mang lại sự hiểu biết sâu sắc hơn về cách chúng có thể được khai thác để giải quyết các thách thức thực tiễn trong quản lý và phân tích dữ liệu. Thông qua dự án này, chúng tôi mong muốn làm rõ những lợi ích và cải tiến đáng kể có thể đạt được trong hiệu suất máy chủ web và trải nghiệm người dùng thông qua việc áp dụng các kỹ thuật xử lý dữ liệu lớn hiện đại.

1. **Triển Khai Dự Án.**

**2.1. Mô Tả Kịch Bản Mô Phỏng.**

Trong dự án này, chúng em mô phỏng quá trình xử lý phức tạp của các tệp log máy chủ web từ Trung tâm Vũ trụ Kennedy của NASA, tập trung vào việc xử lý khối lượng lớn dữ liệu yêu cầu HTTP. Dữ liệu log kéo dài trong khoảng thời gian một tháng, cung cấp một bản ghi chi tiết về các tương tác trên web. Mỗi dòng log bao gồm các thông tin quan trọng như dấu thời gian, loại yêu cầu, mã phản hồi HTTP và kích thước dữ liệu được trao đổi tính bằng byte.

Kịch bản mô phỏng này nhằm chứng minh hiệu quả của các công nghệ dữ liệu lớn trong việc quản lý và phân tích lượng lớn dữ liệu log một cách chính xác và hiệu quả. Thông qua việc triển khai mô phỏng này, chúng tôi mong muốn làm nổi bật một số khía cạnh quan trọng.

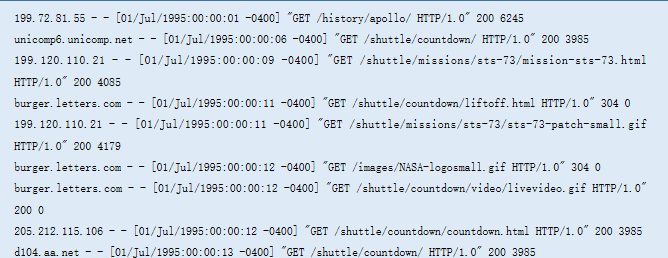
Các log từ Trung tâm Vũ trụ Kennedy của NASA đại diện cho một khối lượng dữ liệu đáng kể, là một môi trường thử nghiệm lý tưởng để đánh giá khả năng xử lý dữ liệu quy mô lớn của hệ thống pipeline mà không ảnh hưởng đến hiệu suất. Các dòng log chi tiết cho phép phân tích chuyên sâu về nhiều khía cạnh khác nhau của các tương tác với máy chủ web, bao gồm việc phân tích sự phân bố mã phản hồi HTTP, nhận diện các mẫu yêu cầu từ phía khách hàng, và đo lường thời gian phản hồi trung bình. Thông qua mô phỏng này, chúng tôi có thể đánh giá nghiêm ngặt hiệu suất và khả năng mở rộng của quy trình xử lý dữ liệu, cũng như xác định các nút thắt cổ chai tiềm ẩn và tối ưu hóa pipeline để đạt hiệu quả cao hơn.

Ngoài ra, việc phân tích tập dữ liệu log lớn này cung cấp các thông tin quan trọng về mô hình sử dụng máy chủ và các chỉ số hiệu suất. Hiểu được các mô hình này sẽ hỗ trợ trong việc tối ưu hóa cấu hình máy chủ, cải thiện trải nghiệm người dùng, và đưa ra các quyết định quản lý dịch vụ web một cách hợp lý. Tóm lại, kịch bản mô phỏng liên quan đến log máy chủ web của Trung tâm Vũ trụ Kennedy cung cấp một khuôn khổ toàn diện để trình bày khả năng của pipeline xử lý dữ liệu lớn. Bằng cách xử lý, phân tích và rút ra thông tin từ tập dữ liệu log quy mô lớn một cách tỉ mỉ, chúng tôi có thể chứng minh những lợi ích thiết thực của việc ứng dụng các công nghệ hiện đại như Hadoop, Flume, Spark và Hive trong các tình huống quản lý và phân tích dữ liệu thực tế.

**2.2. Lựa Chọn và Mô Tả Tập Dữ Liệu**

Trong dự án này, chúng em tập trung phân tích các tệp log máy chủ web từ máy chủ WWW của Trung tâm Vũ trụ Kennedy của NASA, vốn cung cấp một tập dữ liệu phong phú để hiểu rõ các mẫu lưu lượng truy cập web và mức độ sử dụng máy chủ. Tập dữ liệu bao gồm các yêu cầu HTTP trong một tháng, cụ thể là từ ngày 1 tháng 7 năm 1995 đến ngày 28 tháng 7 năm 1995. Tổng cộng, tập dữ liệu chứa khoảng hơn 1.500.000 yêu cầu riêng lẻ, cung cấp một cái nhìn toàn diện về hoạt động của máy chủ trong khoảng thời gian này.

Các tệp log được lưu trữ dưới định dạng ASCII, với mỗi dòng trong tệp log đại diện cho một yêu cầu HTTP duy nhất. Cấu trúc của mỗi dòng log bao gồm một số trường quan trọng: tên máy hoặc địa chỉ IP của khách hàng gửi yêu cầu, dấu thời gian của yêu cầu, phương thức và tài nguyên trong yêu cầu HTTP, mã phản hồi từ máy chủ, và kích thước của phản hồi tính bằng byte. Dấu thời gian được trình bày chi tiết theo định dạng "DAY MON DD HH:MM:YYYY", bao gồm ngày trong tuần, tháng, ngày trong tháng, thời gian theo định dạng 24 giờ, và năm. Mức độ chi tiết này cho phép chúng tôi thực hiện các phân tích theo thời gian cụ thể và sâu sắc hơn về lưu lượng truy cập.



**Hình 2.2 Dữ liệu mẫu.**

Tập dữ liệu này đặc biệt có giá trị vì một số lý do sau:

Thứ nhất, nó ghi lại một khối lượng lớn lưu lượng truy cập web thực tế, cung cấp cơ sở phân tích sát với các tình huống trong thế giới thực.

Thứ hai, dữ liệu bao phủ các thời điểm khác nhau trong ngày và các ngày trong tuần, cho phép chúng tôi xác định các mẫu và xu hướng trong việc sử dụng máy chủ.

Cuối cùng, việc bao gồm nhiều mã phản hồi HTTP và kích thước dữ liệu (byte) giúp chúng tôi không chỉ đánh giá lưu lượng truy cập mà còn phân tích hiệu suất của máy chủ và các vấn đề tiềm ẩn như tỷ lệ lỗi và mức độ tải

Nguồn dữ liệu thuộc Lawrence Berkeley National Laboratory (LBNL) — một trong những phòng thí nghiệm nghiên cứu quốc gia của Bộ Năng lượng Hoa Kỳ: <https://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz>

**2.3. Quy Trình Nạp Dữ Liệu (Data Ingestion Pipeline)**

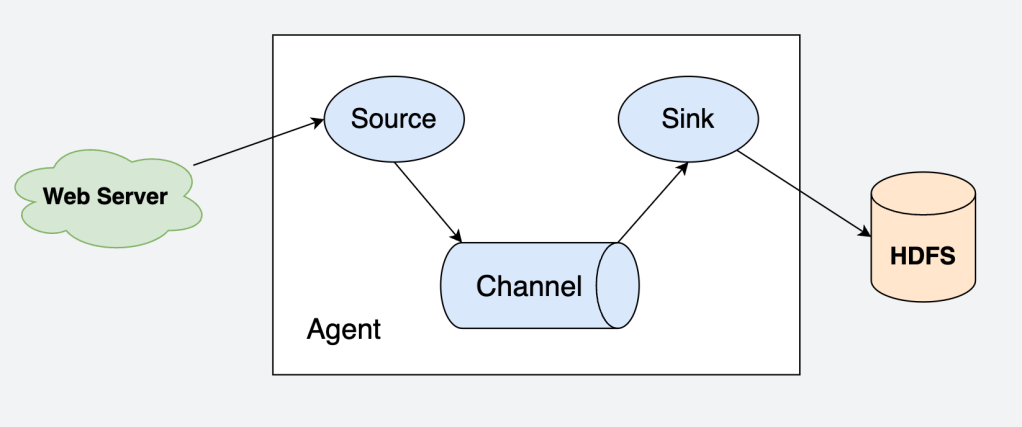
Quy trình nạp dữ liệu được thiết kế để thu thập và lưu trữ hiệu quả dữ liệu log thô vào hệ thống tệp phân tán Hadoop (HDFS) bằng cách sử dụng Apache Flume. Cấu hình của pipeline bao gồm việc thiết lập các tác nhân (agents) Flume với ba thành phần chính: source, channel, và sink để xử lý luồng dữ liệu.

Source (Nguồn): Thành phần này chịu trách nhiệm đọc dữ liệu từ các tệp log đầu vào.

Channel (Kênh trung gian): Dữ liệu sau khi được đọc sẽ được lưu tạm thời trong kênh để đảm bảo tính ổn định và phân tách giữa nguồn và đích.

Sink (Đích): Thành phần này sẽ ghi dữ liệu từ channel vào HDFS.

Cấu trúc này đảm bảo việc nạp dữ liệu diễn ra một cách tin cậy, có khả năng mở rộng cao, và chịu lỗi tốt, đồng thời tạo điều kiện tích hợp dễ dàng với các giai đoạn xử lý dữ liệu tiếp theo như phân tích với Spark hoặc Hive.



**Hình 2.3 Quy trình nạp dữ liệu với Apache Flume.**

Cấu hình này, được minh họa trong **Hình 2.3**, đảm bảo một pipeline nạp dữ liệu **đáng tin cậy**, **có khả năng mở rộng**, và **chịu lỗi tốt**. Nó tạo điều kiện tích hợp liền mạch với các giai đoạn xử lý dữ liệu tiếp theo như **phân tích cú pháp (parsing)**, **làm sạch (cleaning)** và **phân tích (analyzing)** dữ liệu log bằng các công cụ như **PySpark** và **Hive**.

Bằng cách tận dụng Apache Flume và HDFS, pipeline này giải quyết hiệu quả các thách thức trong việc quản lý và phân tích khối lượng lớn dữ liệu log web server. Từ đó, nó giúp đưa ra các quyết định chính xác hơn và tối ưu hóa hiệu suất của máy chủ web.

**2.4. Xử lý Chuyển đổi Dữ liệu (Data Transformation Processing)**

Quá trình chuyển đổi dữ liệu được thực hiện bằng Apache Spark, sử dụng dữ liệu log thô đã được lưu trữ trong Hadoop Distributed File System (HDFS). Quá trình này bao gồm nhiều bước quan trọng nhằm đảm bảo rằng dữ liệu được làm sạch, có cấu trúc rõ ràng, và sẵn sàng cho phân tích.

Làm sạch dữ liệu (Cleaning): Dữ liệu ban đầu được xử lý để loại bỏ các bản ghi không hợp lệ, bị lỗi hoặc không đầy đủ. Đây là bước quan trọng nhằm đảm bảo độ chính xác và chất lượng cho các phân tích sau này.

Phân tích cú pháp (Parsing): Các dòng log sau đó được phân tích để trích xuất các trường quan trọng như: thời gian truy cập (timestamp), phương thức yêu cầu (HTTP method), mã phản hồi (HTTP response code), và kích thước dữ liệu (byte size). Việc phân tích cú pháp giúp chuyển dữ liệu từ dạng văn bản thô sang dạng có cấu trúc.

Tổng hợp dữ liệu (Aggregation): Sau khi đã có dữ liệu có cấu trúc, Spark tiếp tục tổng hợp để rút ra các thông tin có ý nghĩa, chẳng hạn:

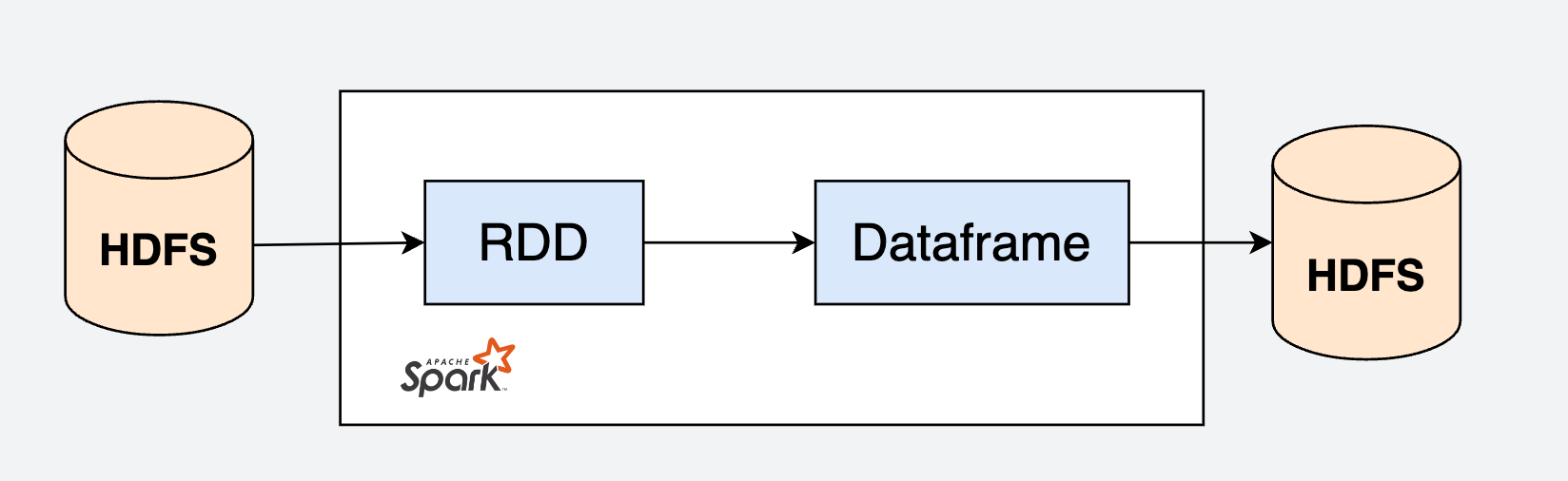
Nhóm theo thời gian để phân tích lưu lượng truy cập theo giờ/ngày/tháng.

Nhóm theo mã phản hồi để đánh giá hiệu suất máy chủ.

Tính trung bình kích thước phản hồi hoặc số lần truy cập từ từng IP.

Khả năng xử lý phân tán mạnh mẽ của Apache Spark giúp toàn bộ quá trình này diễn ra nhanh chóng và hiệu quả, ngay cả khi làm việc với tập dữ liệu có kích thước rất lớn.

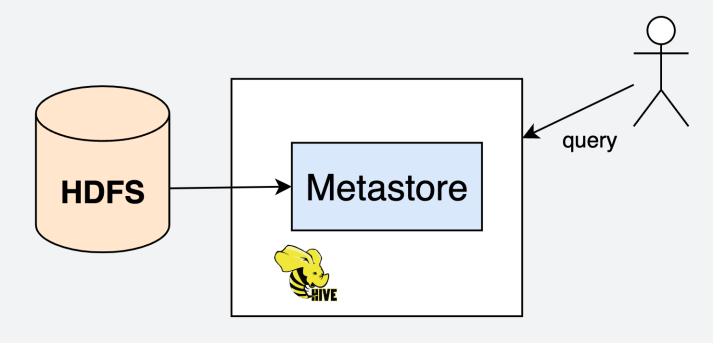
Kết quả là một tập dữ liệu có cấu trúc, sẵn sàng cho việc truy vấn bằng Hive hoặc các công cụ trực quan hóa dữ liệu. Bước này đóng vai trò thiết yếu trong việc chuẩn bị dữ liệu cho việc khai thác thông tin và đưa ra quyết định.



**Hình 2.4 Xử lý dữ liệu với Apache Spark.**

**2.5. Truy vấn Dữ liệu và Khai thác Thông tin (Data Query and Insight)**

Sau khi dữ liệu đã được **chuyển đổi** và **lưu trữ dưới dạng có cấu trúc trong HDFS**, bước tiếp theo là nạp dữ liệu vào **Apache Hive** để thực hiện các truy vấn và phân tích. Hive cung cấp một nền tảng mạnh mẽ để thực hiện các truy vấn tương tự như SQL trên dữ liệu lớn, giúp **trích xuất những thông tin giá trị về hiệu suất và cách sử dụng máy chủ web**.



Hình 2.5 Phân tích dữ liệu với Apache Hive

**2.6. Trực quan hóa dữ liệu (Data Visualization)**

Sau khi dữ liệu đã được truy vấn và các thông tin chi tiết được trích xuất thông qua Apache Hive, bước tiếp theo trong quy trình là trực quan hóa dữ liệu để hỗ trợ việc phân tích và ra quyết định. Trực quan hóa dữ liệu giúp chuyển đổi các con số khô khan thành những biểu đồ, đồ thị dễ hiểu, giúp người dùng nhanh chóng nắm bắt các xu hướng, mối quan hệ và bất thường trong dữ liệu.

Các công cụ như Apache Superset, Tableau hoặc các thư viện trực quan hóa như Matplotlib, Seaborn (Python) hay D3.js (JavaScript) có thể được sử dụng để xây dựng các biểu đồ trực quan từ kết quả truy vấn Hive. Ví dụ, biểu đồ đường (line chart) có thể minh họa lưu lượng truy cập theo thời gian, biểu đồ cột (bar chart) thể hiện phân bố mã phản hồi HTTP, trong khi biểu đồ tròn (pie chart) cho thấy tỷ lệ yêu cầu theo phương thức truy cập (GET, POST, v.v.).

Thông qua các biểu đồ này, người quản trị hệ thống và các nhà phân tích có thể dễ dàng theo dõi hiệu suất của máy chủ web, phát hiện thời điểm lưu lượng cao bất thường, hay xác định nguyên nhân gây ra các lỗi phổ biến. Ngoài ra, việc trực quan hóa cũng hỗ trợ trong việc báo cáo định kỳ, trình bày số liệu với các bên liên quan và ra quyết định chiến lược dựa trên dữ liệu.



Hình 2.6 Các thư viện được dùng trong việc trực quan hóa dữ liệu.

1. **Các Thông Số Cài Đặt**

**3.1 Môi Trường Ảo**

Hệ thống được triển khai trên máy ảo sử dụng VMware Workstation Pro 17.5.

Chi tiết cấu hình máy ảo như sau:

* Hệ điều hành: Ubuntu 22.04.5 (cài đặt từ file ISO ubuntu-22.04.5-desktop-amd64.iso tải trực tiếp từ trang chủ Ubuntu)
* Bộ nhớ RAM: 2GB
* Số lõi xử lý (CPU): 2
* Ổ cứng (SCSI): 100GB
* Kết nối mạng: NAT

**3.2 Hệ Sinh Thái Hadoop**

Phiên bản: Hadoop 3.4.1

Chế độ triển khai: Single-node (đơn nút)

Phiên bản Java: Java 8 (Khuyến nghị sử dụng để đảm bảo khả năng tương thích tốt với Apache Hive trong tương lai)

**3.3 Apache Flume**

Phiên bản: Apache Flume 1.11.0

**3.4 Apache Spark**

Phiên bản: Apache Spark 3.4.4

**3.5 Apache Hive**

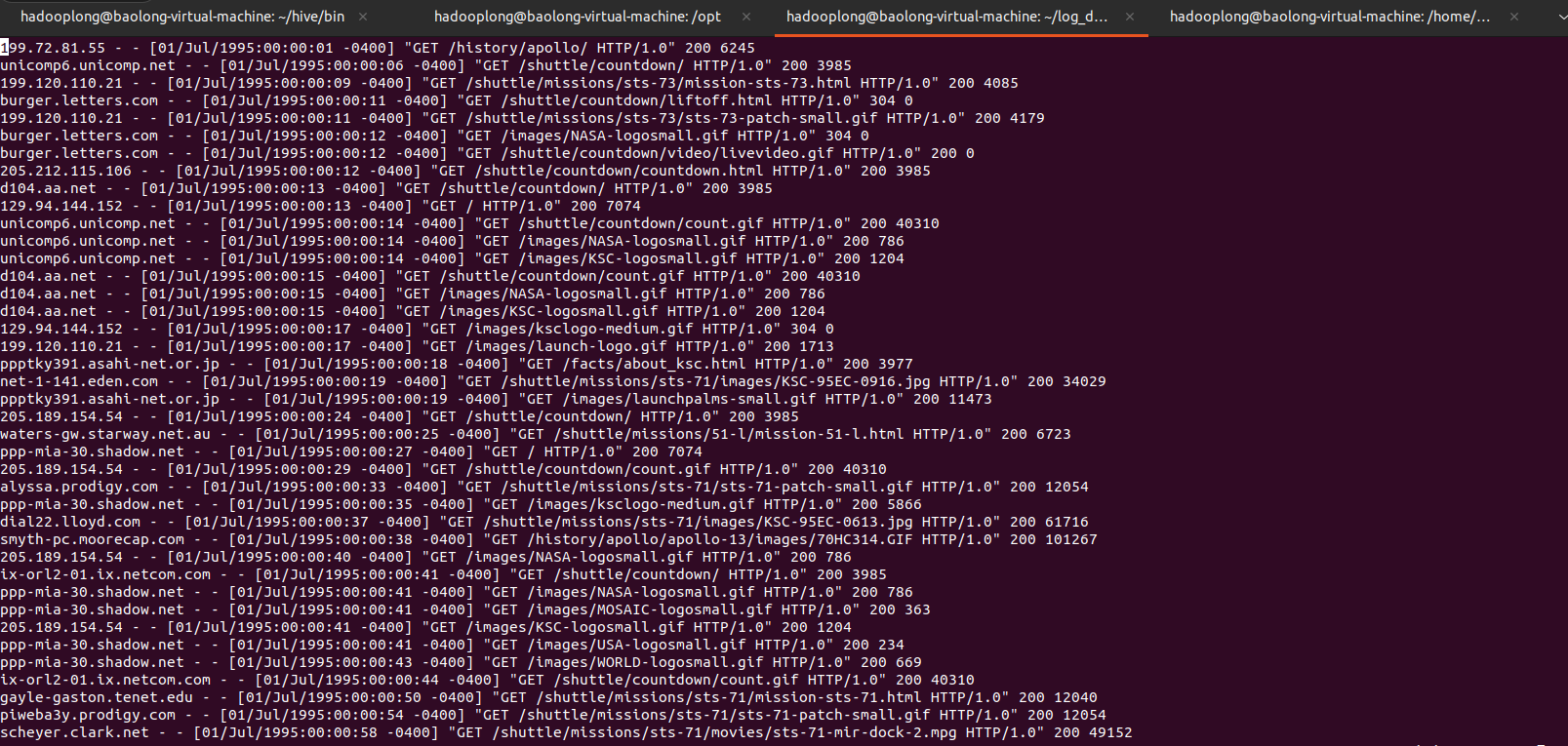
Phiên bản: Apache Hive 3.1.3

1. **Triển Khai Đồ Án.**
   1. **Lấy dữ liệu về từ trang web của phòng thí nghiệm nghiên cứu quốc gia Bộ Năng lượng Hoa Kỳ.**

$ wget <https://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz>

Sau khi tải về và giải nén ta được file dữ liệu lớn.

$ vi NASA\_access\_log\_Jul95



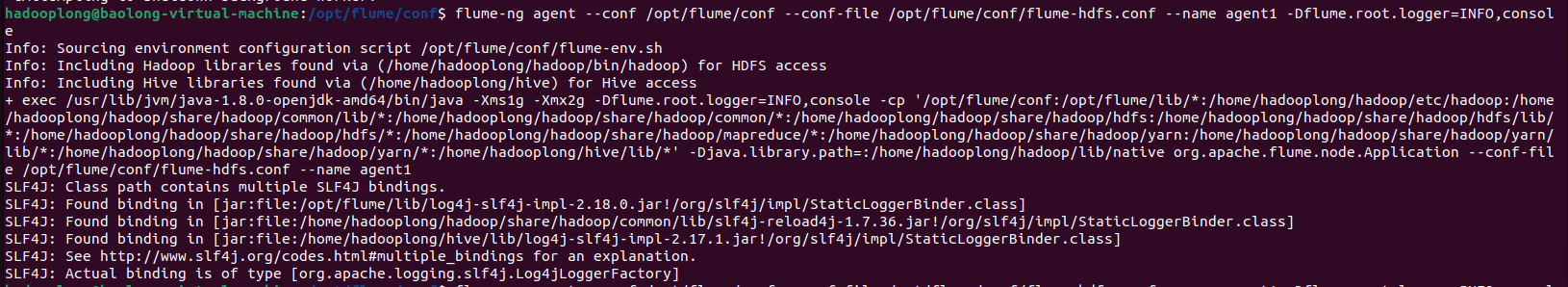
* 1. **Dùng Apache Flume để đẩy dữ liệu lên HDFS (Ứng dụng trong việc truyền dữ liệu trong thời gian thực).**

**$ nano /opt/flume/conf/flume-hdfs.conf**

|  |
| --- |
| # Name the components on this agent  agent.sources = file-source  agent.sinks = hdfs-sink  agent.channels = file-channel  # Associate channel with source and sink  agent.sources.file-source.channels = file-channel  agent.sinks.hdfs-sink.channel = file-channel  # Configure the source  agent.sources.file-source.type = spooldir  agent.sources.file-source.spoolDir = /home/hadooplong/log\_data  agent.sources.file-source.fileHeader = false  agent.sources.file-source.inputCharset = us-ascii  agent.sources.file-source.decodeErrorPolicy = IGNORE  # Configure the sink  agent.sinks.hdfs-sink.type = hdfs  agent.sinks.hdfs-sink.hdfs.path = hdfs://localhost:9000/home/log/data/nasa/july/  agent.sinks.hdfs-sink.hdfs.fileType = DataStream  agent.sinks.hdfs-sink.hdfs.writeFormat = Text  agent.sinks.hdfs-sink.hdfs.rollInterval = 180  agent.sinks.hdfs-sink.hdfs.rollSize = 205242368  agent.sinks.hdfs-sink.hdfs.rollCount = 0  agent.sinks.hdfs-sink.hdfs.batchSize = 1500  agent.sinks.hdfs-sink.hdfs.filePrefix = nasa\_  agent.sinks.hdfs-sink.hdfs.fileSuffix = .log  # Use a channel which buffers events on disk  agent.channels.file-channel.type = file  agent.channels.file-channel.checkpointDir = /home/hadooplong/logprocessing/checkpointDir  agent.channels.file-channel.dataDirs = /home/hadooplong/logprocessing/dataDir |

Khởi chạy Apache Flume:

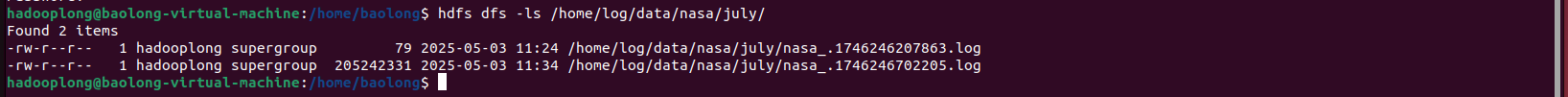
**$**flume-ng agent --conf /opt/flume/conf --conf-file /opt/flume/conf/flume-hdfs.conf --name agent -Dflume.root.logger=INFO,console



Khi này, bất cứ dữ liệu nào được truyền vào /home/hadooplong/log\_data cũng được đẩy lên HDFS.

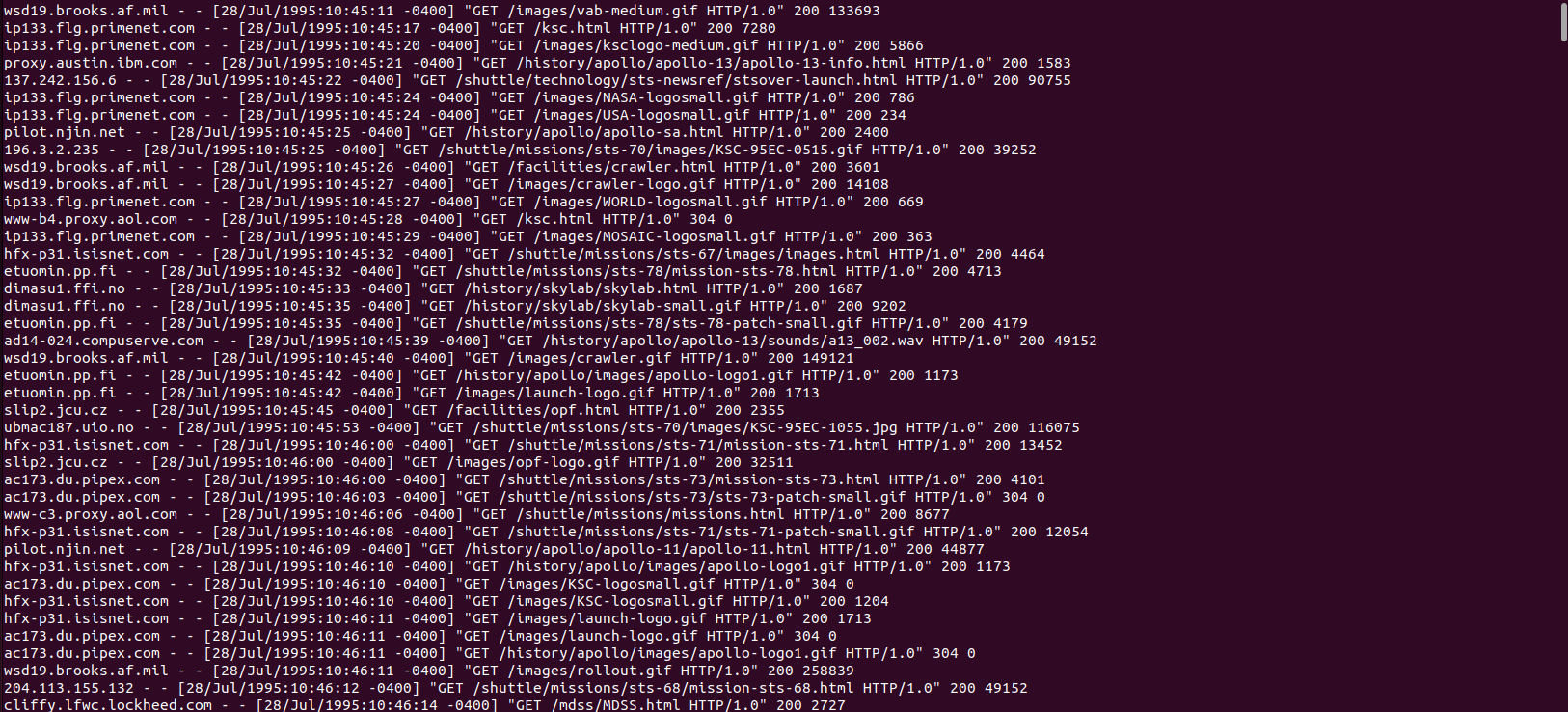
Kiểm tra trên HDFS đã có dữ liệu hay chưa.

**$** hdfs dfs -ls /home/log/data/nasa/july/



**$** hdfs dfs -cat /home/log/data/nasa/july/nasa\_.174624670225





Dữ liệu đã được Apache Flume đẩy lên HDFS thành công.

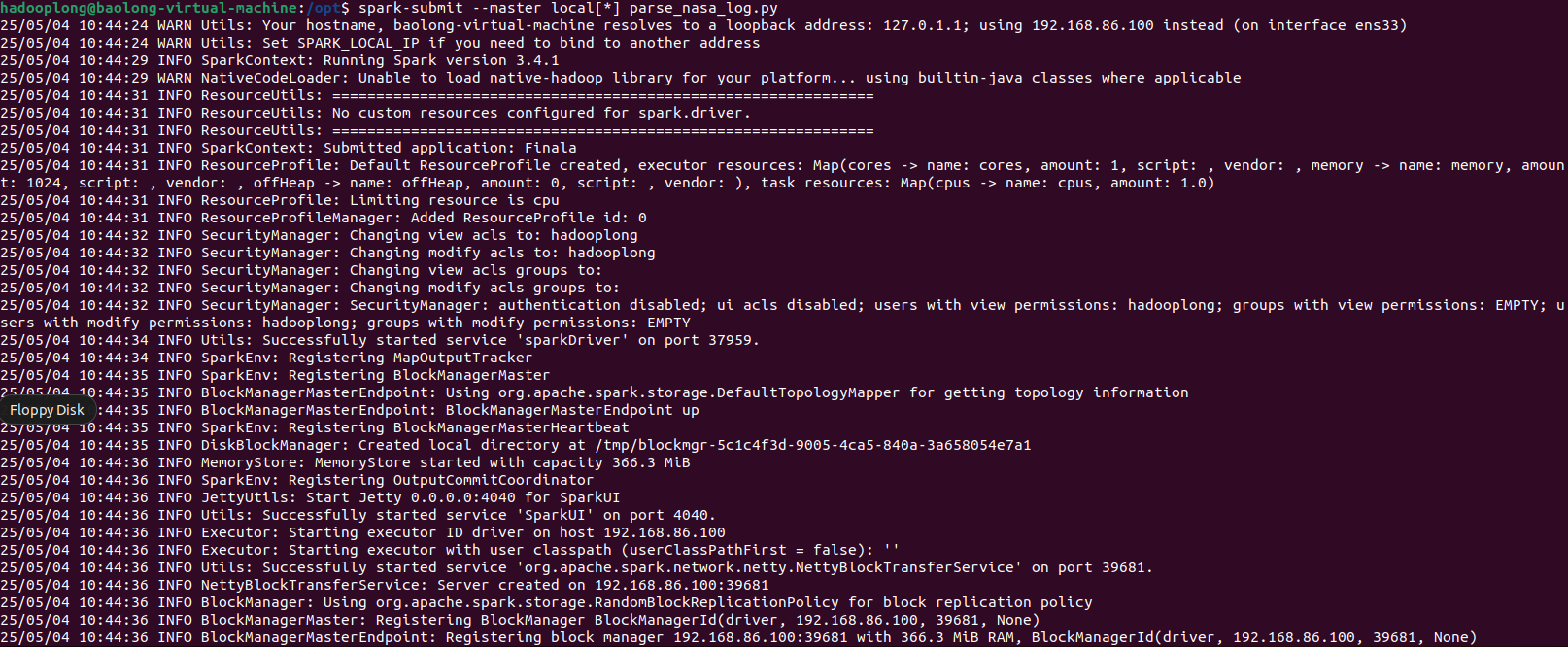
* 1. **Xử lý dữ liệu với Apache Spark.**

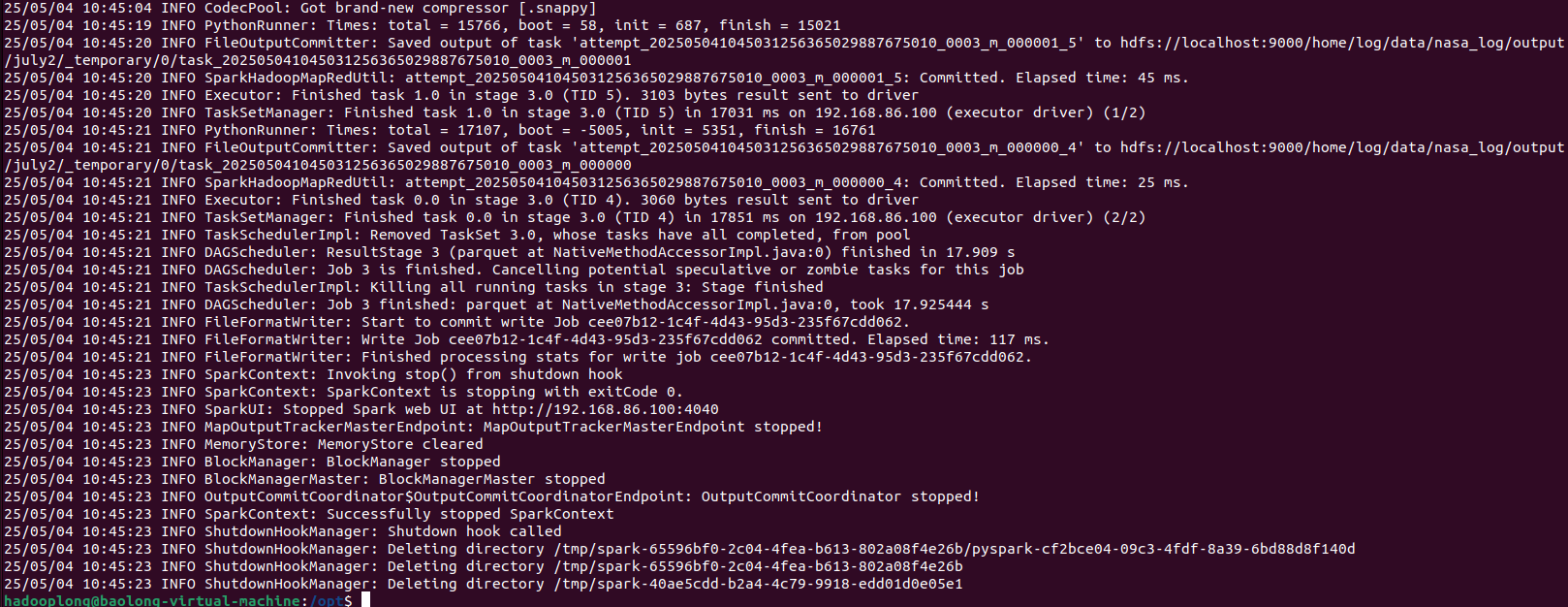
**$** nano /opt/parse\_nasa\_log.py

|  |
| --- |
| from pyspark.sql import SparkSession  from hdfs import InsecureClient  from pyspark.sql.functions import \*  from pyspark.sql.types import \*  import re  # Khởi tạo SparkSession  spark = SparkSession.builder \  .appName("Finala") \  .config("spark.sql.warehouse.dir", "/user/hive/warehouse") \  .config("spark.sql.catalogImplementation", "hive") \  .enableHiveSupport() \  .getOrCreate()  sc = spark.sparkContext  # Kết nối HDFS  client = InsecureClient('http://localhost:9870', user='hadooplong')  hdfs\_path = '/home/log/data/nasa/july/'  file\_list = client.list(hdfs\_path)  # Hàm parse log  def parse\_log(log):  log\_pattern = r'(\S+) - - \[(\d{2}/\w{3}/\d{4}:\d{2}:\d{2}:\d{2}) ([+-]\d{4})\] "(\S+) (\S+) (\S+)" (\d{3}) (\d+)'  match = re.match(log\_pattern, log)  if match:  return match.groups()  else:  return ("", "", "", "", "", "", "", "")  # Xử lý từng file  for file in file\_list:  print(f"Đang xử lý file: {file}")  rdd = sc.textFile(f'hdfs://localhost:9000{hdfs\_path}{file}')  logs\_df = rdd.map(lambda x: parse\_log(x)).toDF([  "host", "timestamp", "tz", "method", "resource", "protocol", "responsecode", "bytes"  ])  # Lọc dữ liệu không hợp lệ  logs\_df = logs\_df.filter(  (col("host") != "") &  (col("timestamp") != "") &  (col("tz") != "") &  (col("method") != "") &  (col("resource") != "") &  (col("protocol") != "") &  (col("responsecode") != "") &  (col("bytes") != "")  )  # Chuyển đổi và thêm các cột thời gian  logs\_df = logs\_df.withColumn("timestamp", to\_timestamp(col("timestamp"), "dd/MMM/yyyy:HH:mm:ss")) \  .withColumn("ts\_year", year("timestamp").cast(IntegerType())) \  .withColumn("ts\_month", month("timestamp").cast(IntegerType())) \  .withColumn("ts\_day", dayofmonth("timestamp").cast(IntegerType())) \  .withColumn("ts\_hour", hour("timestamp").cast(IntegerType())) \  .withColumn("ts\_minute", minute("timestamp").cast(IntegerType())) \  .withColumn("ts\_sec", second("timestamp").cast(IntegerType())) \  .withColumn("ts\_dayOfWeek", dayofweek("timestamp").cast(IntegerType())) \  .withColumn("clientAuthId", col("host")) \  .withColumn("userId", col("host"))  # Ghi ra định dạng parquet  logs\_df.write.mode("append").parquet("/home/log/data/nasa\_log/output/july2") |

Khởi chạy Apache Spark.

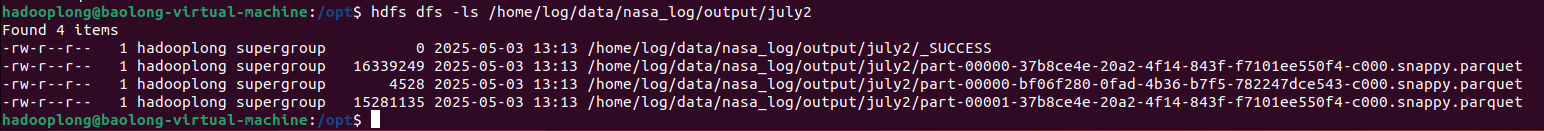
**$** spark-submit --master local[\*] parse\_nasa\_log.py





Sau khi Apache Spark chạy xong kiểm tra trên hdfs đã có dữ liệu sau khi xử lý chưa.

**$** hdfs dfs -ls /home/log/data/nasa\_log/output/july2



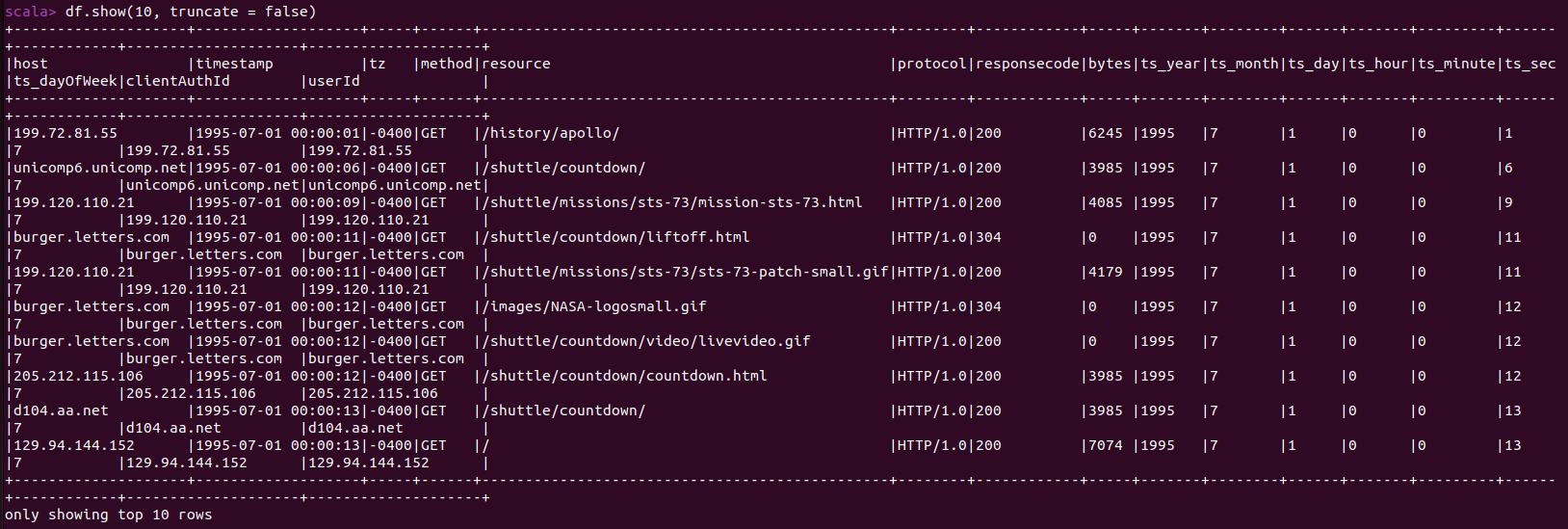
Sau khi xử lý, dữ liệu được chia nhỏ thành 3 file và lưu dưới dạng parquet file.

Để xem được nội dung của file này, ta cần vào spark-shell để đọc.

$ spark-shell

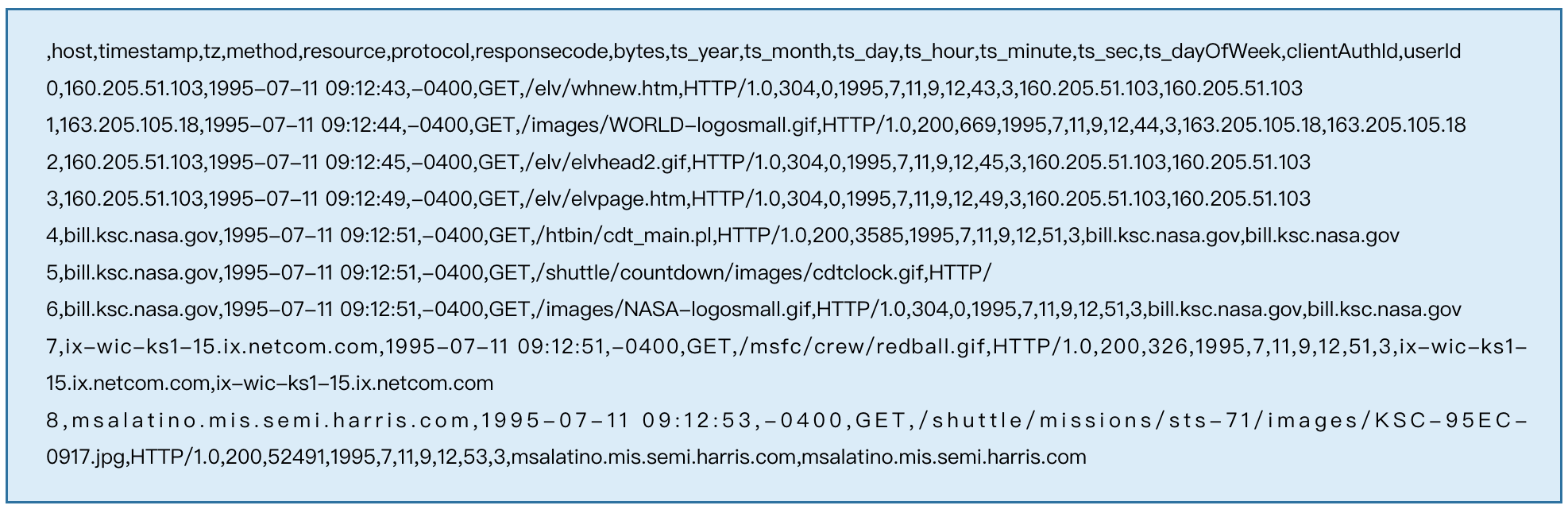
**scala>** val df = spark.read.parquet("/home/log/data/nasa\_log/output/july2")

**scala>** df.show(10, truncate = false)



Dữ liệu đã được xử lý thành các trường khác nhau.

Dưới đây là dữ liệu đã qua xử lý:



Mô tả các trường dữ liệu.

|  |  |  |
| --- | --- | --- |
| **Column name** | **Description** | **Sample** |
| **host** | The hostname or IP address of the client making the request. | dd05-029.compuserve.com |
| **timestamp** | The date and time when the request was made. | 1995-07-20 20:14:41 |
| **tz** | The time zone of the server. | -0400 |
| **method** | The HTTP method used for the request (e.g., GET). | GET |
| **resource** | The requested resource URL. | /history/apollo/as-201/sounds/ |
| **protocol** | The HTTP protocol used (e.g., HTTP/1.0). | HTTP/1.0 |
| **responsecode** | The HTTP response code indicating the result of the request. | 200 |
| **bytes** | The size of the response in bytes. | 372 |
| **ts\_year** | The year of the request. | 1995 |
| **ts\_month** | The month of the request. | 7 |
| **ts\_day** | The day of the month of the request. | 20 |
| **ts\_hour** | The hour of the request. | 20 |
| **ts\_minute** | The minute of the request. | 14 |
| **ts\_sec** | The second of the request. | 1 |
| **ts\_dayOfWeek** | The day of the week of the request. | 5 |
| **clientAuthId** | An identifier for the client. | dd05-029.compuserve.com |
| **userId** | The ID of the user making the request. | dd05-029.compuserve.com |

**Giải thích chi tiết:**

**Host**: Cột này ghi lại tên máy (hostname) hoặc địa chỉ IP của client (khách hàng) đã gửi yêu cầu. Thông tin này rất quan trọng để xác định nguồn gốc lưu lượng truy cập và có thể được sử dụng cho các phân tích sâu hơn như theo dõi vị trí địa lý hoặc phân tích hành vi người dùng.

**Timestamp**: Dấu thời gian ghi lại chính xác ngày và giờ khi yêu cầu được gửi. Đây là yếu tố thiết yếu trong phân tích theo thời gian, giúp hiểu rõ các mô hình truy cập và thời điểm truy cập cao điểm.

**Time Zone (tz)**: Múi giờ của máy chủ giúp điều chỉnh dấu thời gian về thời gian chuẩn (UTC) hoặc so sánh log từ các máy chủ ở múi giờ khác nhau.

**HTTP Method (method)**: Cột này chỉ ra phương thức HTTP được sử dụng trong yêu cầu, như GET, POST, PUT, v.v. Điều này quan trọng để hiểu loại thao tác mà client đang thực hiện trên máy chủ.

**Resource**: URL của tài nguyên được yêu cầu cho biết trang hoặc tập tin cụ thể nào đã được client truy cập. Thông tin này giúp xác định những tài nguyên phổ biến để tối ưu hiệu năng của chúng.

**Protocol**: Phiên bản giao thức HTTP được sử dụng trong yêu cầu. Việc biết rõ giao thức giúp đánh giá khả năng tương thích và các yếu tố bảo mật của yêu cầu.

**Response Code**: Mã phản hồi HTTP chỉ ra kết quả của yêu cầu, như thành công (200), không tìm thấy (404), hoặc lỗi máy chủ (500). Việc phân tích các mã này giúp phát hiện sự cố và cải thiện độ ổn định của máy chủ.

**Bytes**: Cột này ghi lại kích thước phản hồi tính bằng byte. Nó hữu ích cho việc phân tích băng thông và đánh giá mức độ tải của máy chủ.

**Temporal Attributes (ts\_year, ts\_month, ts\_day, ts\_hour, ts\_minute, ts\_sec, ts\_dayOfWeek)**: Các cột này phân tách dấu thời gian thành các thành phần riêng biệt, cho phép phân tích dữ liệu một cách chi tiết hơn. Ví dụ, có thể truy vấn để tìm khung giờ truy cập cao nhất hoặc ngày trong tuần có lưu lượng lớn nhất.

**ClientAuthId và UserId**: Hai cột này cung cấp định danh cho client và người dùng. Chúng có thể được sử dụng để theo dõi hành vi người dùng, phân tích phiên làm việc, và xác định người dùng duy nhất.

-> Cấu trúc chi tiết này cho phép phân tích nhật ký máy chủ web một cách hiệu quả và toàn diện, mang lại những hiểu biết giá trị về mô hình sử dụng và hiệu suất hoạt động của máy chủ.

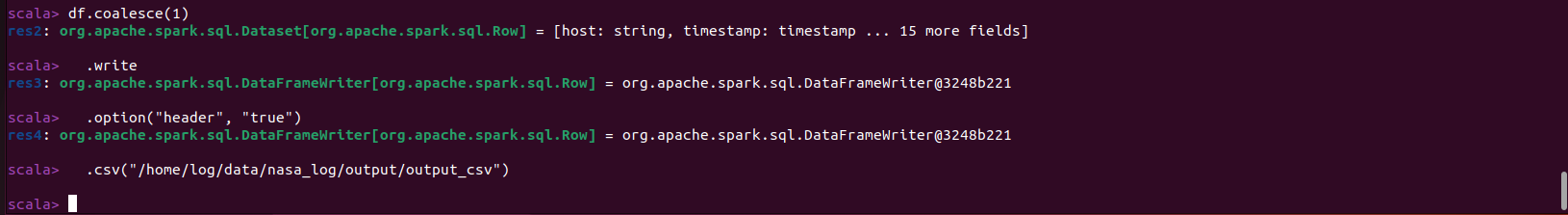
Tuy nhiên, hiện tại dữ liệu được lưu trữ dưới dạng file parquet rất khó để sử dụng map-reduce hay truy vấn nên cần chuyển sang một dạng khác là file csv để dễ dàng làm việc.

**scala>** df.coalesce(1)

.write

.option("header", "true")

.csv("/home/log/data/nasa\_log/output/output\_csv")



Kiểm tra trên hdfs:

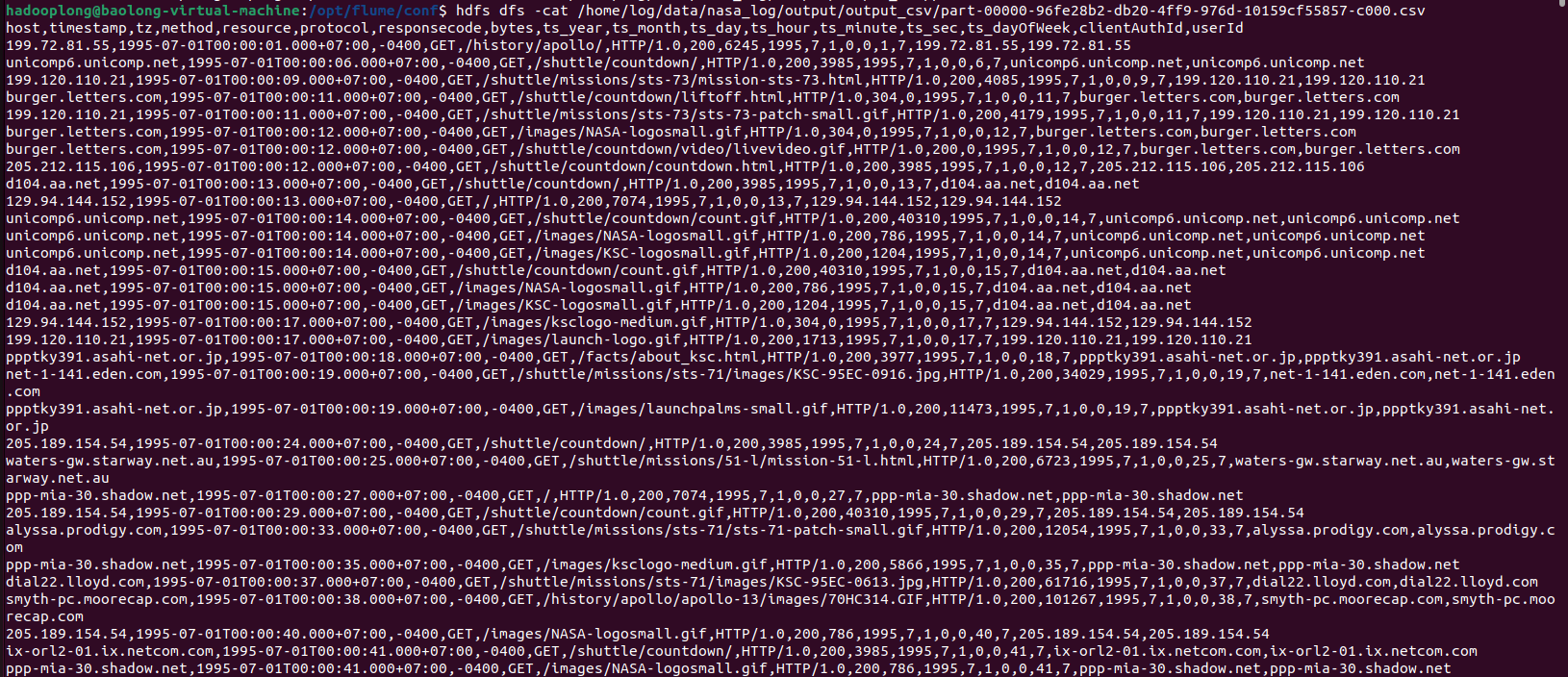
**$** hdfs dfs -ls /home/log/data/nasa\_log/output/output\_csv



Đã có file csv trong thư mục, kiểm tra nội dung trong file.

|  |
| --- |
| host,timestamp,tz,method,resource,protocol,responsecode,bytes,ts\_year,ts\_month,ts\_day,ts\_hour,ts\_minute,ts\_sec,ts\_dayOfWeek,clientAuthId,userId  199.72.81.55,1995-07-01T00:00:01.000+07:00,-0400,GET,/history/apollo/,HTTP/1.0,200,6245,1995,7,1,0,0,1,7,199.72.81.55,199.72.81.55  unicomp6.unicomp.net,1995-07-01T00:00:06.000+07:00,-0400,GET,/shuttle/countdown/,HTTP/1.0,200,3985,1995,7,1,0,0,6,7,unicomp6.unicomp.net,unicomp6.unicomp.net  199.120.110.21,1995-07-01T00:00:09.000+07:00,-0400,GET,/shuttle/missions/sts-73/mission-sts-73.html,HTTP/1.0,200,4085,1995,7,1,0,0,9,7,199.120.110.21,199.120.110.21  burger.letters.com,1995-07-01T00:00:11.000+07:00,-0400,GET,/shuttle/countdown/liftoff.html,HTTP/1.0,304,0,1995,7,1,0,0,11,7,burger.letters.com,burger.letters.com  199.120.110.21,1995-07-01T00:00:11.000+07:00,-0400,GET,/shuttle/missions/sts-73/sts-73-patch-small.gif,HTTP/1.0,200,4179,1995,7,1,0,0,11,7,199.120.110.21,199.120.110.21  burger.letters.com,1995-07-01T00:00:12.000+07:00,-0400,GET,/images/NASA-logosmall.gif,HTTP/1.0,304,0,1995,7,1,0,0,12,7,burger.letters.com,burger.letters.com  burger.letters.com,1995-07-01T00:00:12.000+07:00,-0400,GET,/shuttle/countdown/video/livevideo.gif,HTTP/1.0,200,0,1995,7,1,0,0,12,7,burger.letters.com,burger.letters.com  205.212.115.106,1995-07-01T00:00:12.000+07:00,-0400,GET,/shuttle/countdown/countdown.html,HTTP/1.0,200,3985,1995,7,1,0,0,12,7,205.212.115.106,205.212.115.106  d104.aa.net,1995-07-01T00:00:13.000+07:00,-0400,GET,/shuttle/countdown/,HTTP/1.0,200,3985,1995,7,1,0,0,13,7,d104.aa.net,d104.aa.net  129.94.144.152,1995-07-01T00:00:13.000+07:00,-0400,GET,/,HTTP/1.0,200,7074,1995,7,1,0,0,13,7,129.94.144.152,129.94.144.152  unicomp6.unicomp.net,1995-07-01T00:00:14.000+07:00,-0400,GET,/shuttle/countdown/count.gif,HTTP/1.0,200,40310,1995,7,1,0,0,14,7,unicomp6.unicomp.net,unicomp6.unicomp.net  unicomp6.unicomp.net,1995-07-01T00:00:14.000+07:00,-0400,GET,/images/NASA-logosmall.gif,HTTP/1.0,200,786,1995,7,1,0,0,14,7,unicomp6.unicomp.net,unicomp6.unicomp.net  unicomp6.unicomp.net,1995-07-01T00:00:14.000+07:00,-0400,GET,/images/KSC-logosmall.gif,HTTP/1.0,200,1204,1995,7,1,0,0,14,7,unicomp6.unicomp.net,unicomp6.unicomp.net  d104.aa.net,1995-07-01T00:00:15.000+07:00,-0400,GET,/shuttle/countdown/count.gif,HTTP/1.0,200,40310,1995,7,1,0,0,15,7,d104.aa.net,d104.aa.net |

**$** hdfs dfs -cat /home/log/data/nasa\_log/output/output\_csv/part-00000-96fe28b2-db20-4ff9-976d-10159cf55857-c000.csv



Từ sau chúng ta sẽ xử lý trên file csv một cách dễ dàng

* 1. **Đưa dữ liệu vào Apache Hive và truy vấn dữ liệu.**

CREATE EXTERNAL TABLE nasa\_logs (

host STRING,

`timestamp` STRING,

tz STRING,

method STRING,

resource STRING,

protocol STRING,

responsecode INT,

bytes INT,

ts\_year INT,

ts\_month INT,

ts\_day INT,

ts\_hour INT,

ts\_minute INT,

ts\_sec INT,

ts\_dayOfWeek STRING,

clientAuthId STRING,

userId STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

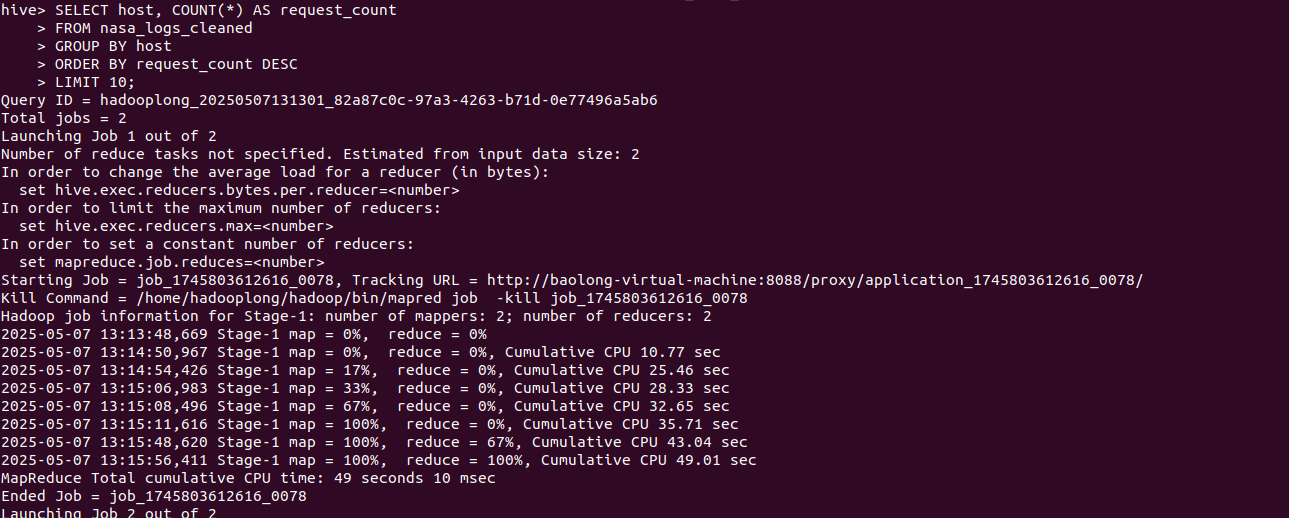
STORED AS TEXTFILE

LOCATION '/home/log/data/nasa\_log/output/output\_csv/';

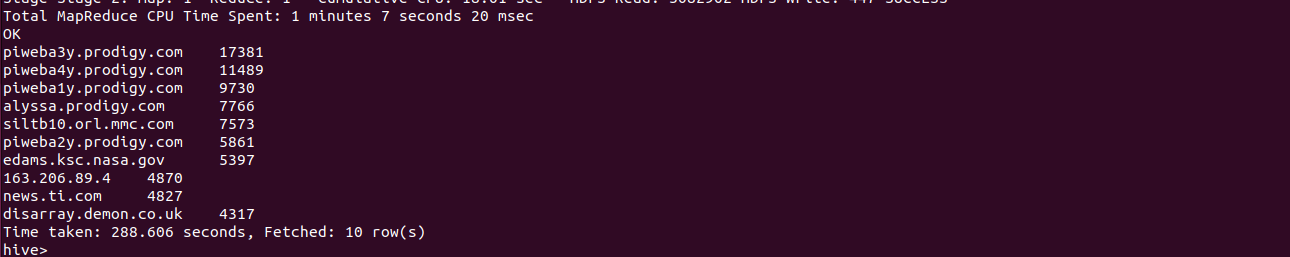


Top 10 khách hàng request nhiều nhất.

|  |
| --- |
| SELECT host, COUNT(\*) AS request\_count  FROM nasa\_logs\_cleaned  GROUP BY host  ORDER BY request\_count DESC  LIMIT 10; |

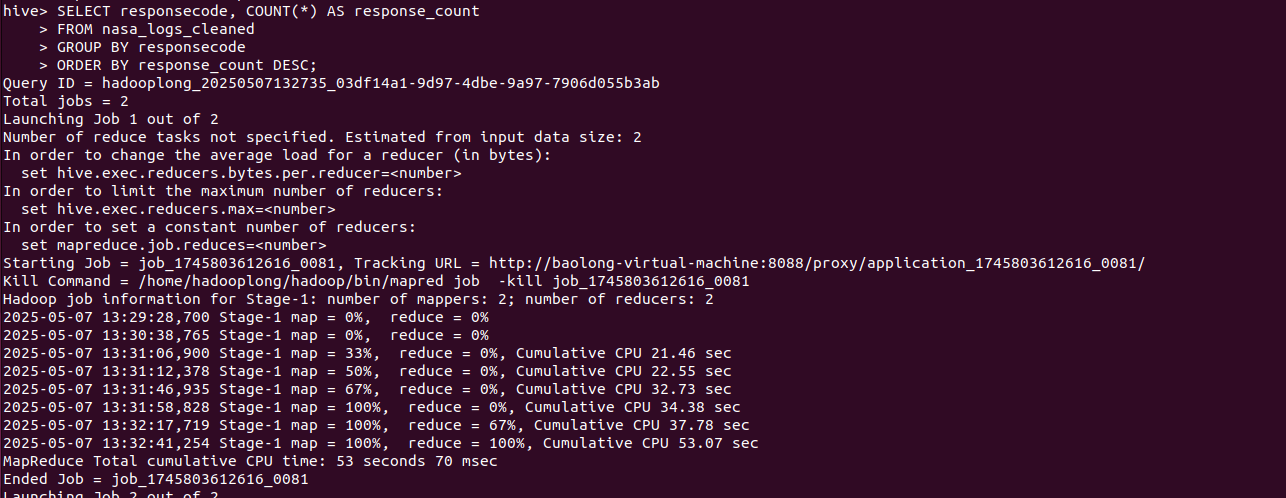


Kết quả:



Phân phối mã phản hồi HTTP

|  |
| --- |
| SELECT responsecode, COUNT(\*) AS response\_count  FROM nasa\_logs\_cleaned  GROUP BY responsecode  ORDER BY response\_count DESC; |

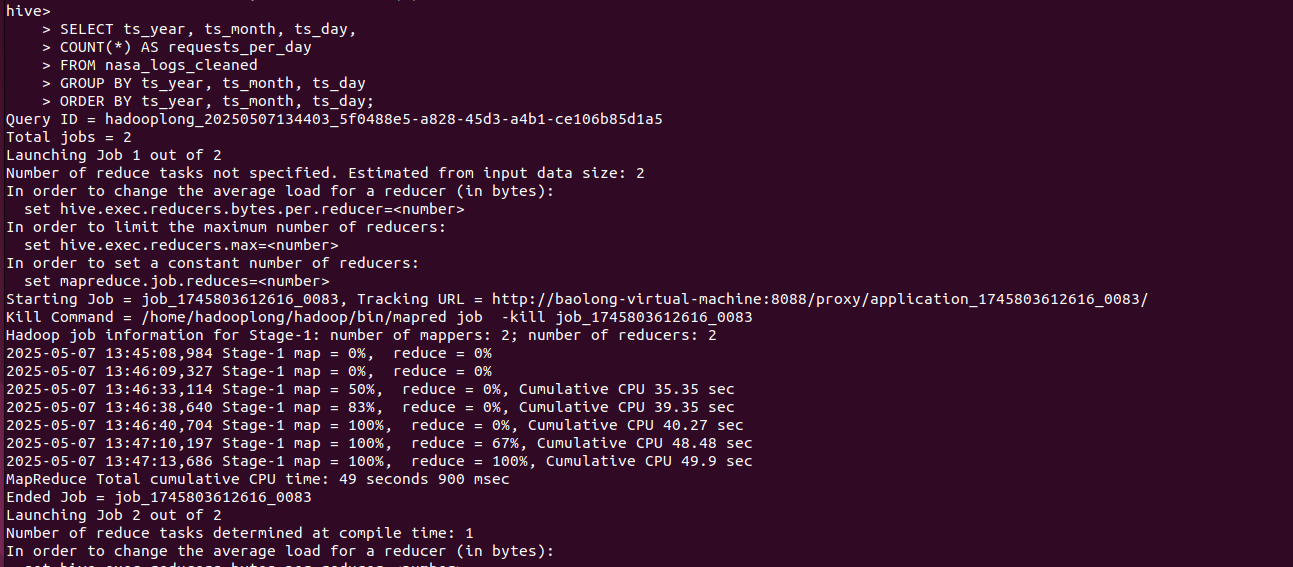


Kết quả:



Số lượng request mỗi ngày.

|  |
| --- |
| SELECT ts\_year, ts\_month, ts\_day,  COUNT(\*) AS requests\_per\_day  FROM nasa\_logs\_cleaned  GROUP BY ts\_year, ts\_month, ts\_day  ORDER BY ts\_year, ts\_month, ts\_day; |



Kết quả:

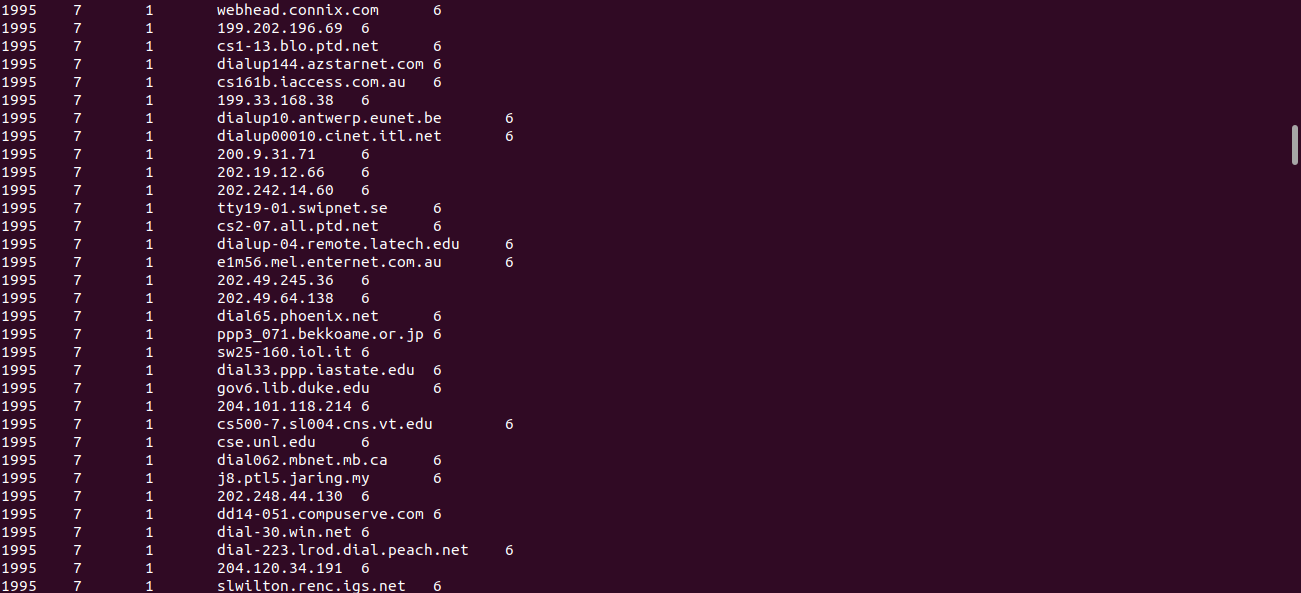


Số lượng request của khách hàng theo thời gian.

|  |
| --- |
| SELECT ts\_year, ts\_month, ts\_day, host, COUNT(\*) AS request\_count  FROM nasa\_logs\_cleaned  GROUP BY ts\_year, ts\_month, ts\_day, host  ORDER BY ts\_year, ts\_month, ts\_day, request\_count DESC; |



Kết quả:



* 1. **Sử dụng Hadoop Streaming để tiến hành Map-reduce.**
     1. **Đếm số lần xuất hiện của từng host.**

mapper1.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  for line in sys.stdin:  if line.startswith("host") or line.strip() == "":  continue  parts = line.split(",")  print(f"{parts[0]}\t1") |

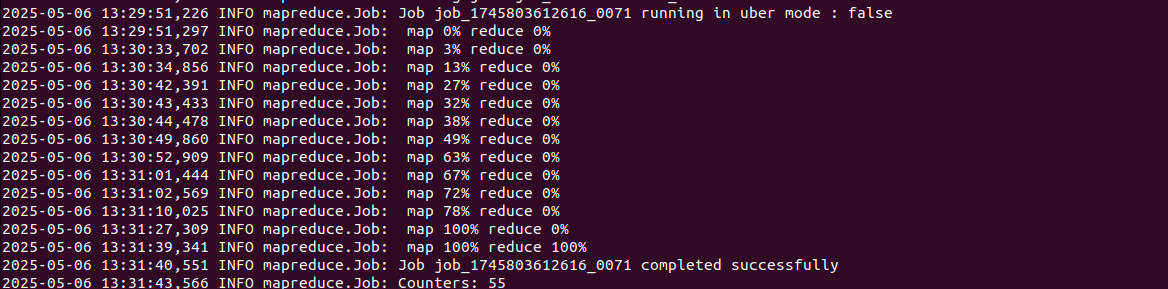
reducer1.py

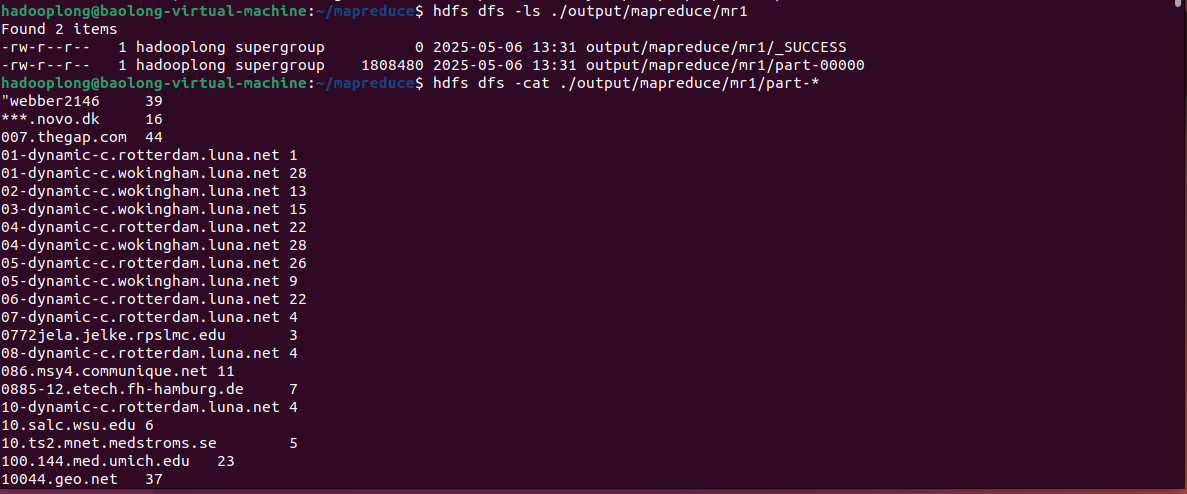
|  |
| --- |
| #!/usr/bin/env python3  import sys  from collections import defaultdict  counts = defaultdict(int)  for line in sys.stdin:  host, count = line.strip().split("\t")  counts[host] += int(count)  for host, total in counts.items():  print(f"{host}\t{total}") |

Chạy map-reduce:

$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -input /home/log/data/nasa\_log/output/output\_csv/part-\*.csv -output ./output/mapreduce/mr1 -mapper mapper1.py -reducer reducer1.py -file mapper1.py -file reducer1.py

Kết quả:





…



* + 1. **Đếm số request theo mã phản hồi HTTP (response code).**

mapper2.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  for line in sys.stdin:  if line.startswith("code") or line.strip() == "":  continue  parts = line.split(",")  print(f"{parts[6]}\t1") |

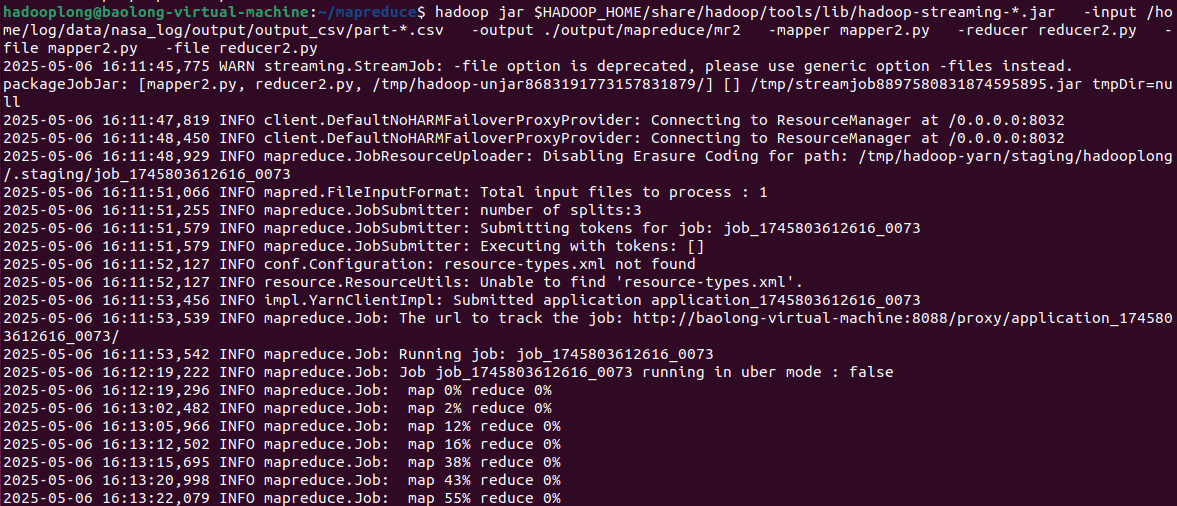
reducer2.py

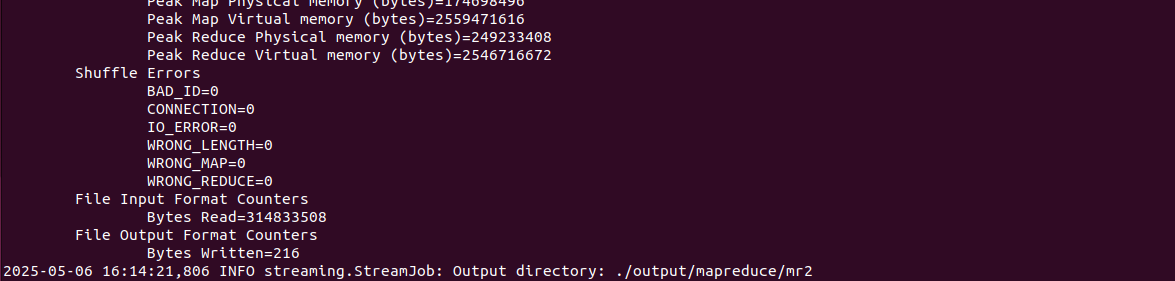
|  |
| --- |
| #!/usr/bin/env python3  import sys  from collections import defaultdict  counts = defaultdict(int)  for line in sys.stdin:  code, count = line.strip().split("\t")  counts[code] += int(count)  for code, total in counts.items():  print(f"{code}\t{total}") |

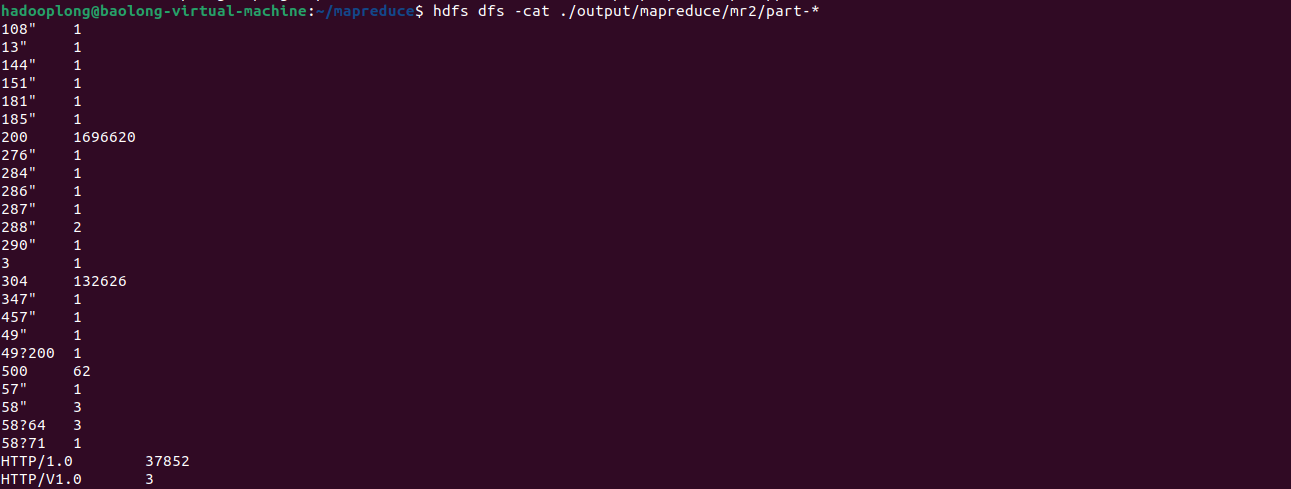
Chạy map-reduce:

$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -input /home/log/data/nasa\_log/output/output\_csv/part-\*.csv -output ./output/mapreduce/mr2 -mapper mapper2.py -reducer reducer2.py -file mapper2.py -file reducer2.py

Kết quả:







* + 1. **Tổng số bytes trả về.**

mapper3.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  for line in sys.stdin:  if line.startswith("host"):  continue  try:  bytes\_sent = int(line.split(",")[6])  print(f"total\t{bytes\_sent}")  except:  continue |

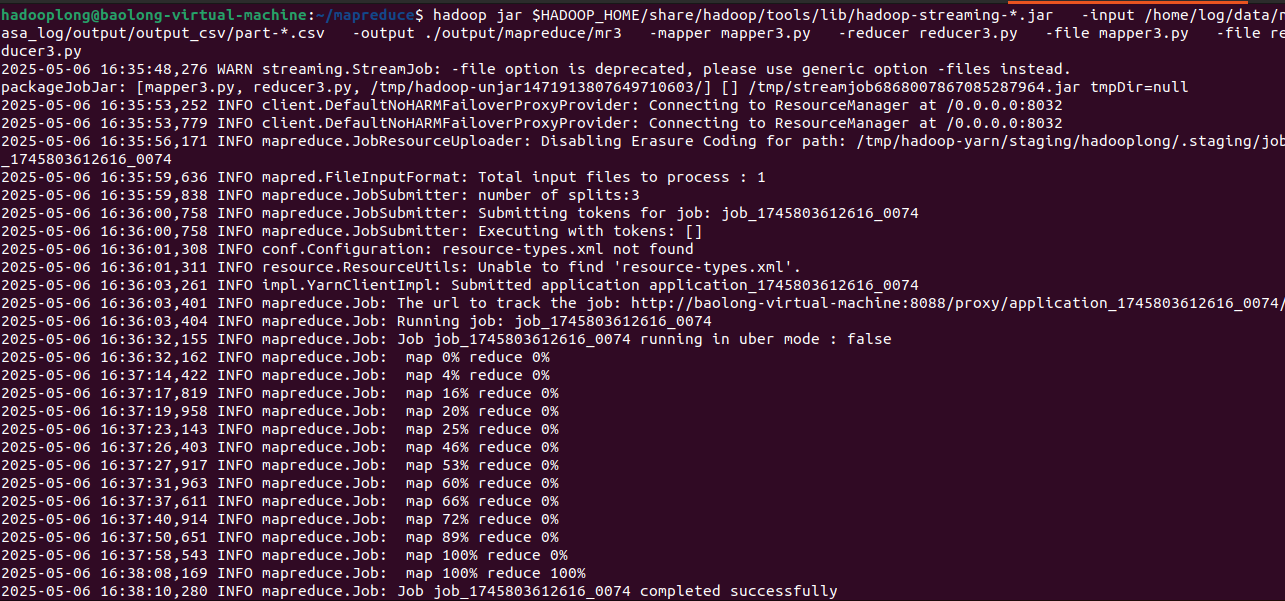
reducer3.py

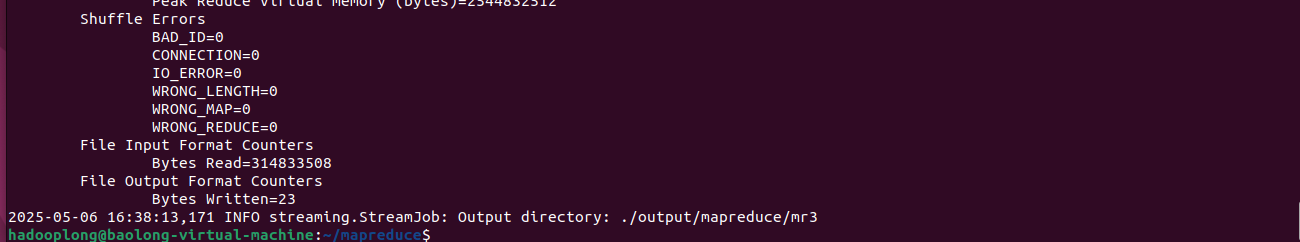
|  |
| --- |
| #!/usr/bin/env python3  import sys  total = 0  for line in sys.stdin:  \_, val = line.strip().split("\t")  total += int(val)  print(f"Total Bytes:\t{total}") |

Chạy map-reduce:

$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -input /home/log/data/nasa\_log/output/output\_csv/part-\*.csv -output ./output/mapreduce/mr3 -mapper mapper3.py -reducer reducer3.py -file mapper3.py -file reducer3.py

Kết quả:







* + 1. **Tính tổng lượt truy cập theo giờ.**

mapper4.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  import csv  for line in sys.stdin:  if line.startswith("host"):  continue  try:  fields = next(csv.reader([line]))  hour = fields[11] # Cột ts\_hour  print(f"{hour}\t1")  except:  continue |

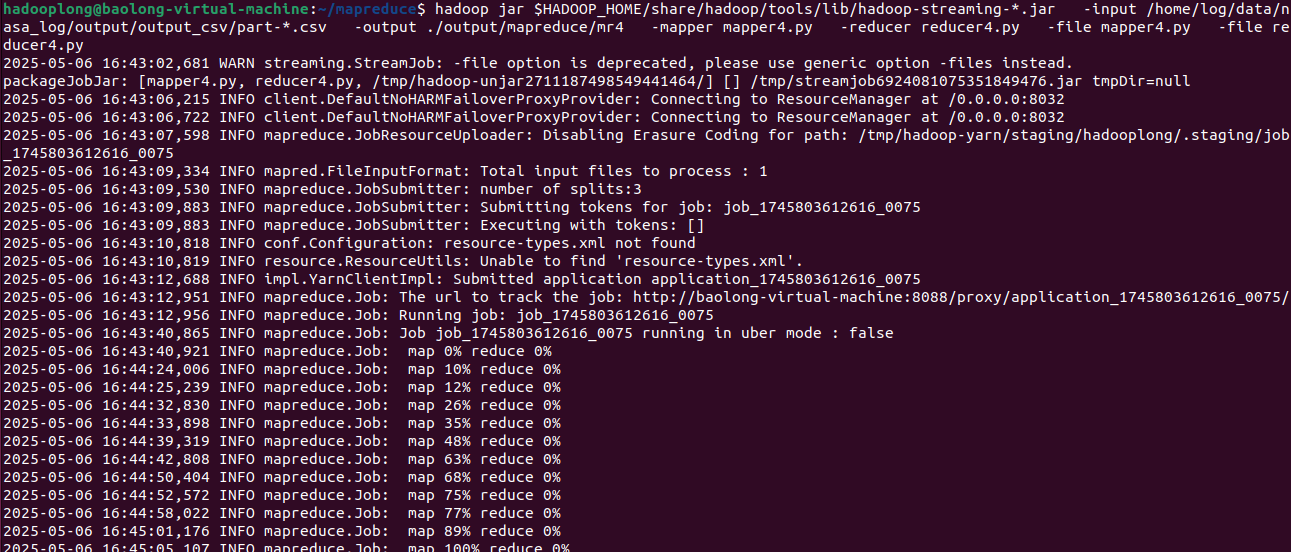
reducer4.py

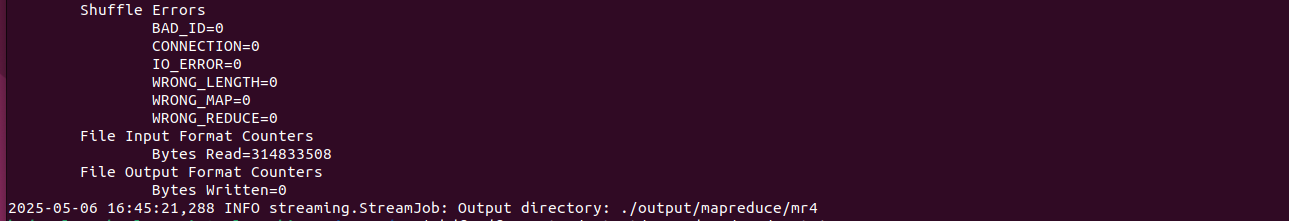
|  |
| --- |
| #!/usr/bin/env python3  import sys  current\_code = None  count = 0  for line in sys.stdin:  code, val = line.strip().split("\t")  val = int(val)  if code == current\_code:  count += val  else:  if current\_code:  print(f"{current\_code}\t{count}")  current\_code = code  count = val  if current\_code:  print(f"{current\_code}\t{count}") |

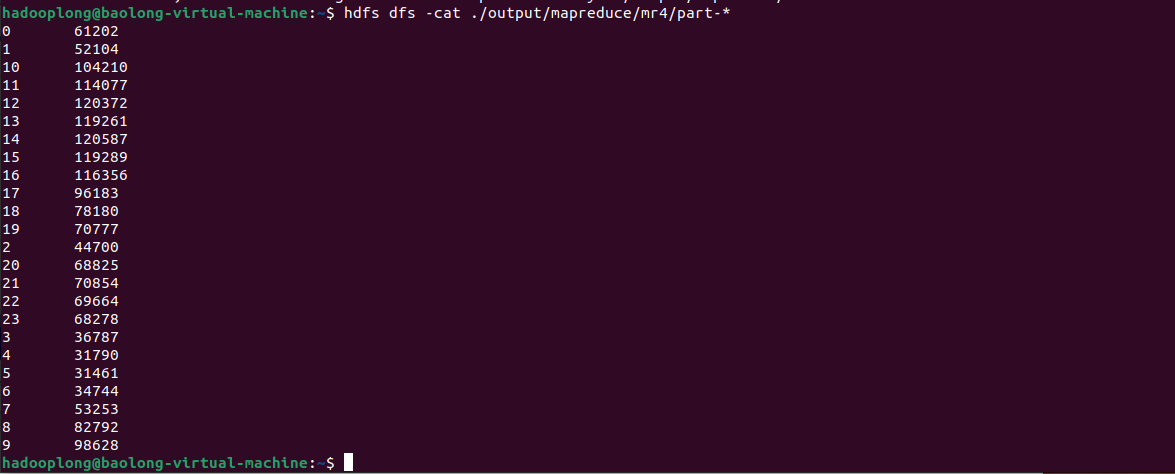
Chạy map-reduce:

$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -input /home/log/data/nasa\_log/output/output\_csv/part-\*.csv -output ./output/mapreduce/mr4 -mapper mapper4.py -reducer reducer4.py -file mapper4.py -file reducer4.py

Kết quả:







* + 1. **Số lượng truy cập theo nguồn tài nguyên.**

mapper5.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  import csv  for line in sys.stdin:  try:  fields = next(csv.reader([line]))  resource = fields[4]  print(f"{resource}\t1")  except:  continue |

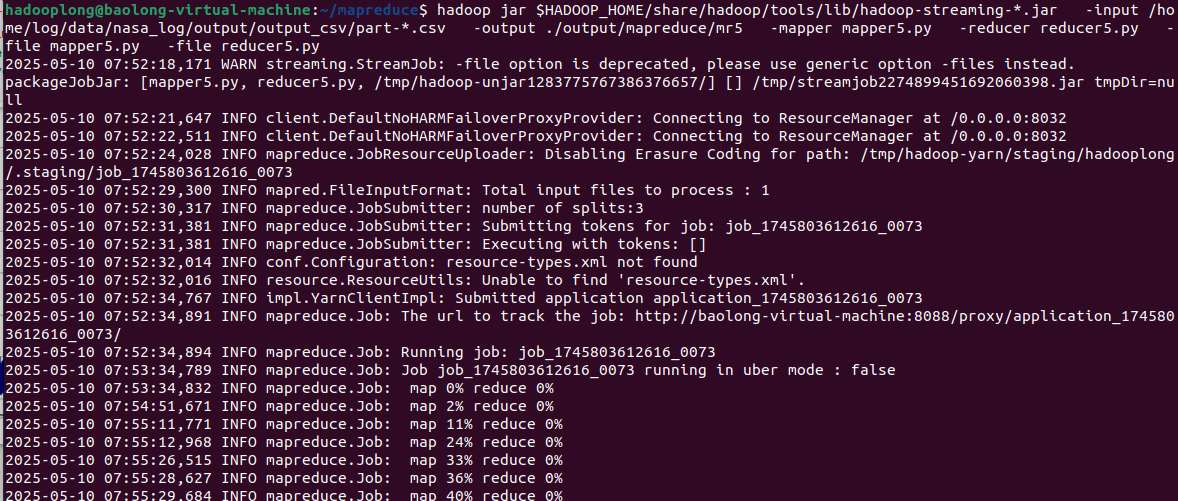
reducer5.py

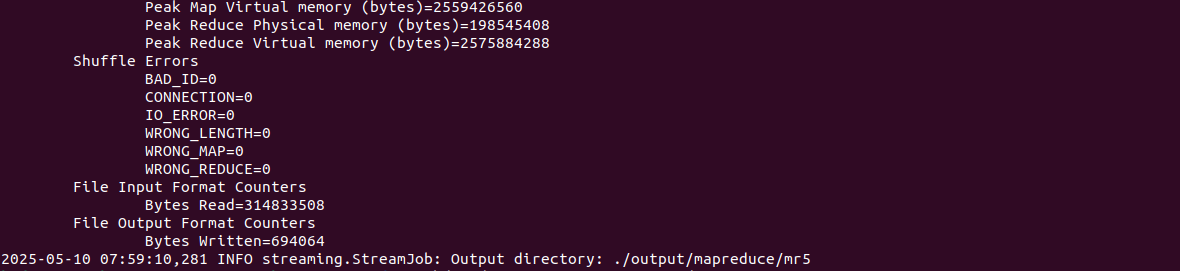
|  |
| --- |
| #!/usr/bin/env python3  import sys  current\_code = None  count = 0  for line in sys.stdin:  code, val = line.strip().split("\t")  val = int(val)  if code == current\_code:  count += val  else:  if current\_code:  print(f"{current\_code}\t{count}")  current\_code = code  count = val  if current\_code:  print(f"{current\_code}\t{count}") |

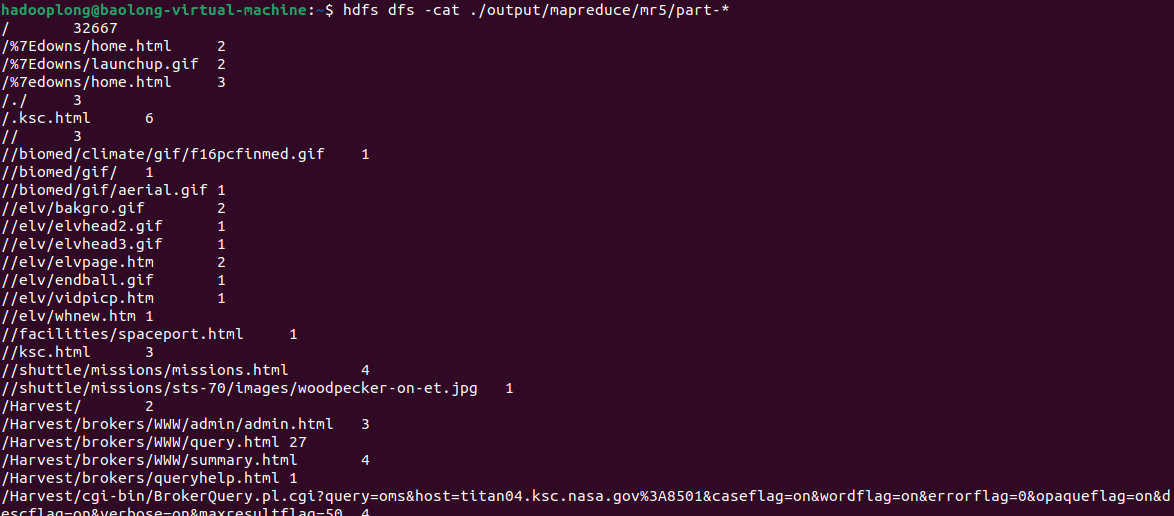
Chạy map-reduce:

$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -input /home/log/data/nasa\_log/output/output\_csv/part-\*.csv -output ./output/mapreduce/mr5 -mapper mapper5.py -reducer reducer5.py -file mapper5.py -file reducer5.py

Kết quả:







* + 1. **Top 5 host truy cập nhiều nhất theo dung lượng.**

mapper6.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  import csv  reader = csv.reader(sys.stdin)  header = next(reader, None)  if header and header[0] != "host":  reader = csv.reader([",".join(header)] + list(sys.stdin))  for row in reader:  try:  host = row[0]  bytes\_sent = int(row[6]) if row[6] != '-' else 0  print(f"{host}\t{bytes\_sent}")  except Exception as e:  continue |

reducer6.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  from collections import defaultdict  host\_bytes = defaultdict(int)  for line in sys.stdin:  host, val = line.strip().split("\t")  host\_bytes[host] += int(val)  top\_hosts = sorted(host\_bytes.items(), key=lambda x: x[1], reverse=True)[:5]  for host, total in top\_hosts:  print(f"{host}\t{total}") |

Chạy map-reduce:

$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -input /home/log/data/nasa\_log/output/output\_csv/part-\*.csv -output ./output/mapreduce/mr6 -mapper mapper6.py -reducer reducer6.py -file mapper6.py -file reducer6.py

Kết quả:



* + 1. **Tính tổng lượng truy cập theo mỗi IP/Hostname vào nguồn resource có tiền tố URL /image.**

mapper7.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  import csv  for line in sys.stdin:  try:  row = next(csv.reader([line]))  host = row[0].strip()  resource = row[4].strip() # Cột 'resource'  if resource.startswith("/image"):  print(f"{host}\t1")  except Exception:  continue # Bỏ qua dòng lỗi |

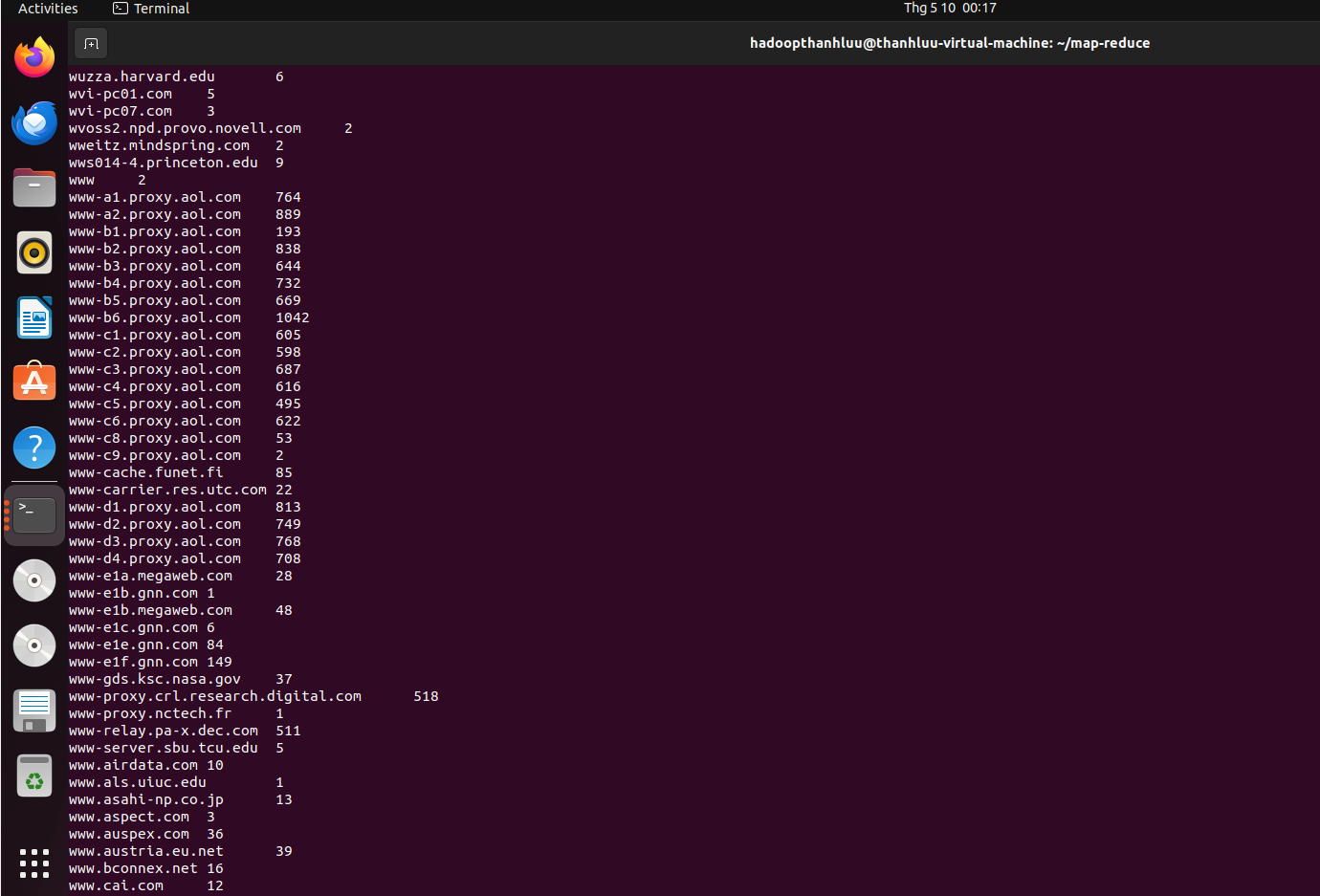
reducer7.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  current\_host = None  current\_count = 0  for line in sys.stdin:  try:  host, count = line.strip().split('\t')  count = int(count)  if current\_host == host:  current\_count += count  else:  if current\_host is not None:  print(f"{current\_host}\t{current\_count}")  current\_host = host  current\_count = count  except:  continue  # In kết quả cuối cùng  if current\_host:  print(f"{current\_host}\t{current\_count}") |

Chạy map-reduce:

$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -input /home/log/data/nasa\_log/output/output\_csv/part-\*.csv -output ./output/mapreduce/mr7 -mapper mapper7.py -reducer reducer7.py -file mapper7.py -file reducer7.py

Kết quả:



* + 1. **Tính tổng số lần truy cập theo loại URL.**

mapper8.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  import csv  for line in sys.stdin:  try:  row = next(csv.reader([line]))  resource = row[4].strip() # Cột 'resource'  if resource.startswith("/"):  # Lấy phần loại tài nguyên (prefix sau dấu '/')  parts = resource.split('/')  if len(parts) > 1 and parts[1]:  resource\_type = parts[1].split('/')[0]  print(f"{resource\_type}\t1")  except Exception:  continue |

reducer8.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  current\_type = None  current\_count = 0  for line in sys.stdin:  try:  resource\_type, count = line.strip().split('\t')  count = int(count)  if current\_type == resource\_type:  current\_count += count  else:  if current\_type is not None:  print(f"{current\_type}\t{current\_count}")  current\_type = resource\_type  current\_count = count  except:  continue  # In loại cuối cùng  if current\_type:  print(f"{current\_type}\t{current\_count}") |

Chạy map-reduce:

$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -input /home/log/data/nasa\_log/output/output\_csv/part-\*.csv -output ./output/mapreduce/mr8 -mapper mapper8.py -reducer reducer8.py -file mapper8.py -file reducer8.py

Kết quả:



* + 1. **Tính số lượng truy cập theo mã trạng thái HTTP như 200, 500,…**

mapper9.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  import csv  for line in sys.stdin:  try:  row = next(csv.reader([line]))  status\_code = row[6].strip() # Cột 'responsecode'  if status\_code:  print(f"{status\_code}\t1")  except Exception:  continue |

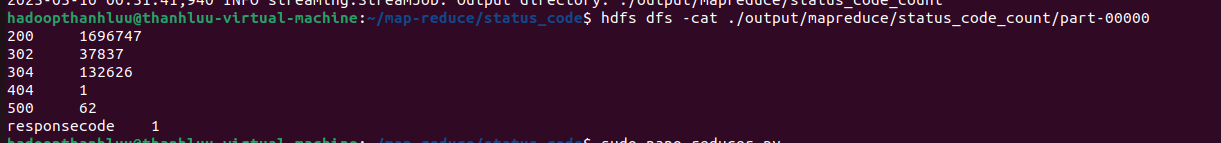
reducer9.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  current\_code = None  current\_count = 0  for line in sys.stdin:  try:  code, count = line.strip().split('\t')  count = int(count)  if current\_code == code:  current\_count += count  else:  if current\_code is not None:  print(f"{current\_code}\t{current\_count}")  current\_code = code  current\_count = count  except:  continue |

Chạy map-reduce:

$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -input /home/log/data/nasa\_log/output/output\_csv/part-\*.csv -output ./output/mapreduce/mr9 -mapper mapper9.py -reducer reducer9.py -file mapper9.py -file reducer9.py

Kết quả:



* + 1. **Tìm loại tài nguyên có lượt truy cập nhiều nhất.**

rapper10.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  for line in sys.stdin:  try:  resource\_type, count = line.strip().split('\t')  print(f"max\t{resource\_type},{count}")  except:  continue |

reducer10.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  max\_type = None  max\_count = 0  for line in sys.stdin:  try:  \_, value = line.strip().split('\t')  resource\_type, count = value.split(',')  count = int(count)  if count > max\_count:  max\_count = count  max\_type = resource\_type  except:  continue  if max\_type:  print(f"{max\_type}\t{max\_count}") |

Chạy map-reduce:

$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -input /home/log/data/nasa\_log/output/output\_csv/part-\*.csv -output ./output/mapreduce/mr10 -mapper mapper10.py -reducer reducer10.py -file mapper10.py -file reducer10.py

Kết quả:



* + 1. **Tính số byte trung bình trả về theo từng URL.**

mapper11.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  import csv  def extract\_type(resource):  parts = resource.strip().split('/')  if len(parts) > 1 and parts[1]:  return parts[1].lower() # lấy phần sau dấu "/" đầu tiên  return "unknown"  reader = csv.reader(sys.stdin)  header = next(reader) # bỏ qua header  for row in reader:  try:  resource = row[4] # cột resource  bytes\_sent = row[7] # cột bytes  if not resource:  continue  resource\_type = extract\_type(resource)  bytes\_val = int(bytes\_sent) if bytes\_sent.isdigit() else 0  print(f"{resource\_type}\t{bytes\_val}\t1") # emit: type, bytes, count  except Exception:  continue |

reducer11.py

|  |
| --- |
| #!/usr/bin/env python3  import sys  current\_type = None  total\_bytes = 0  total\_count = 0  for line in sys.stdin:  try:  resource\_type, bytes\_val, count = line.strip().split("\t")  bytes\_val = int(bytes\_val)  count = int(count)  if current\_type != resource\_type:  if current\_type:  avg = total\_bytes / total\_count if total\_count else 0  print(f"{current\_type}\t{avg:.2f}")  current\_type = resource\_type  total\_bytes = bytes\_val  total\_count = count  else:  total\_bytes += bytes\_val  total\_count += count  except:  continue  # In kết quả cuối cùng  if current\_type:  avg = total\_bytes / total\_count if total\_count else 0  print(f"{current\_type}\t{avg:.2f}") |

Chạy map-reduce:

$ hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar -input /home/log/data/nasa\_log/output/output\_csv/part-\*.csv -output ./output/mapreduce/mr11 -mapper mapper11.py -reducer reducer11.py -file mapper11.py -file reducer11.py

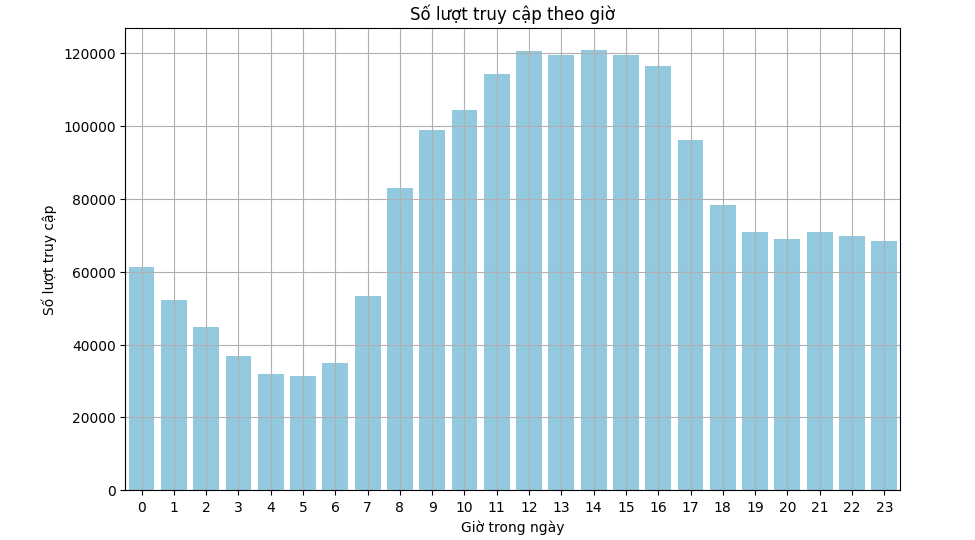
Kết quả:



* 1. **Trực quan hóa dữ liệu bằng python.**
     1. **Số lượng truy cập theo giờ.**

|  |
| --- |
| # Chuyển timestamp về dạng datetime nếu chưa  df['timestamp'] = pd.to\_datetime(df['timestamp'])  # Thêm cột giờ  df['hour'] = df['timestamp'].dt.hour  # Đếm truy cập theo giờ  hourly\_counts = df['hour'].value\_counts().sort\_index()  # Vẽ biểu đồ  plt.figure(figsize=(10, 6))  sns.barplot(x=hourly\_counts.index, y=hourly\_counts.values, color='skyblue')  plt.title("Số lượt truy cập theo giờ")  plt.xlabel("Giờ trong ngày")  plt.ylabel("Số lượt truy cập")  plt.xticks(range(0, 24))  plt.grid(True)  plt.show() |

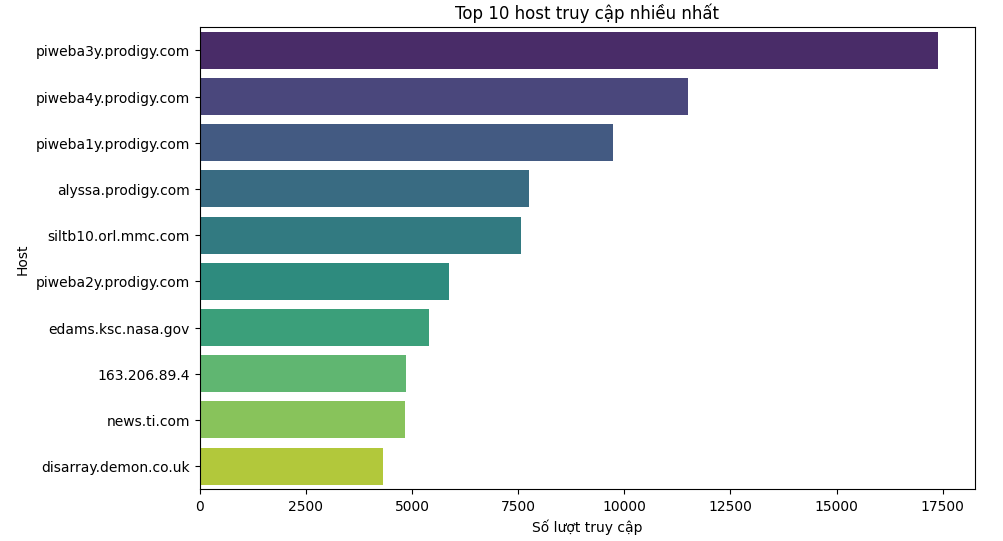
Kết quả:



* + 1. **Top 10 host truy cập nhiều nhất.**

|  |
| --- |
| top\_hosts = df['host'].value\_counts().head(10)  plt.figure(figsize=(10, 6))  sns.barplot(x=top\_hosts.values, y=top\_hosts.index, palette="viridis")  plt.title("Top 10 host truy cập nhiều nhất")  plt.xlabel("Số lượt truy cập")  plt.ylabel("Host")  plt.show() |

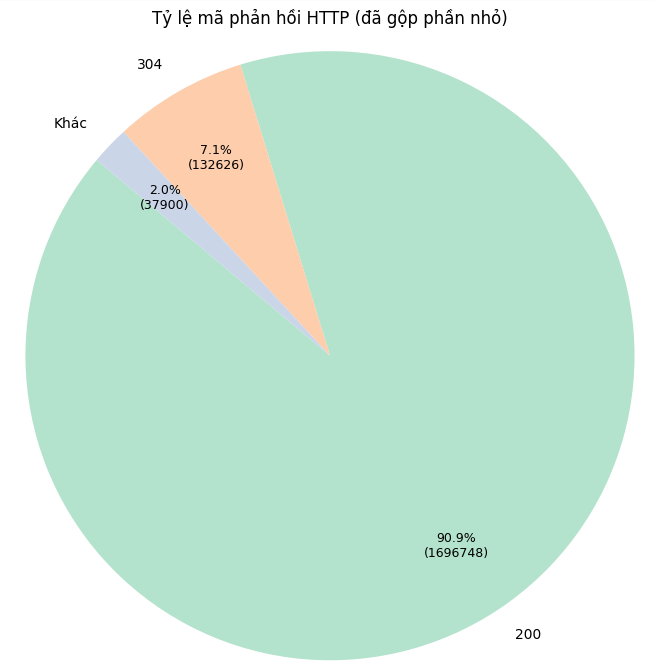
Kết quả:



* + 1. **Tỷ lệ các mã phản hồi (Response Code).**

|  |
| --- |
| response\_counts = df['responsecode'].value\_counts()  total = response\_counts.sum()  # Gộp những mã có tỷ lệ < 3% vào 'Khác'  threshold = 0.03  grouped = response\_counts[response\_counts / total >= threshold]  others = response\_counts[response\_counts / total < threshold].sum()  if others > 0:  grouped['Khác'] = others  labels = grouped.index.astype(str)  sizes = grouped.values  def make\_autopct(values):  def my\_autopct(pct):  total = sum(values)  val = int(round(pct \* total / 100.0))  return f'{pct:.1f}%\n({val})'  return my\_autopct  plt.figure(figsize=(7, 7))  wedges, texts, autotexts = plt.pie(  sizes,  labels=labels,  autopct=make\_autopct(sizes),  startangle=140,  colors=plt.cm.Pastel2.colors,  labeldistance=1.1,  pctdistance=0.75  )  for text in texts:  text.set\_size(10)  for autotext in autotexts:  autotext.set\_size(9)  plt.title("Tỷ lệ mã phản hồi HTTP (đã gộp phần nhỏ)")  plt.axis('equal')  plt.tight\_layout()  plt.show() |

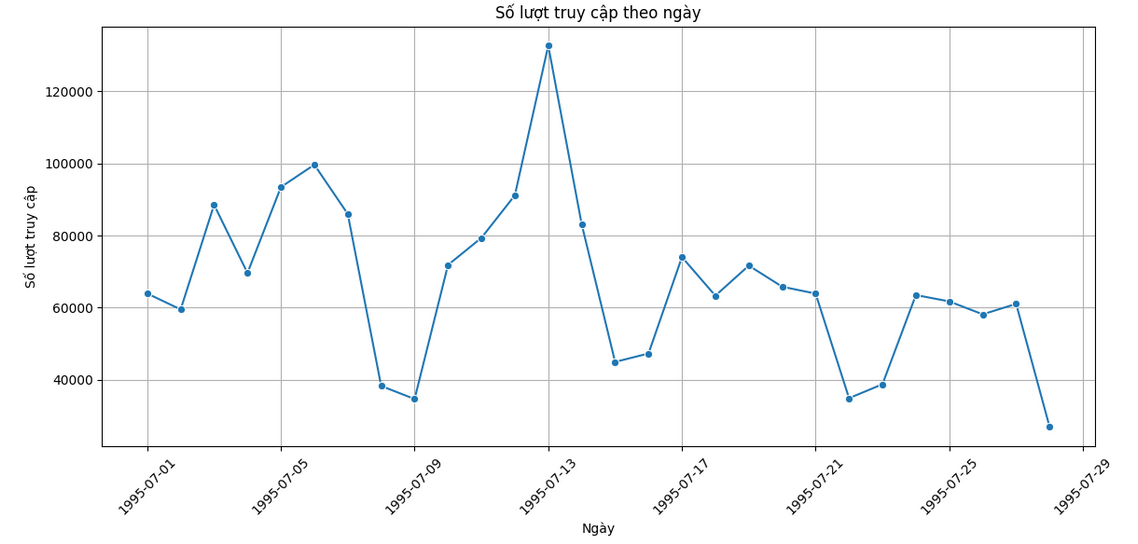
Kết quả:



* + 1. **Số lượt truy cập theo ngày.**

|  |
| --- |
| df['date'] = df['timestamp'].dt.date  daily\_counts = df['date'].value\_counts().sort\_index()  plt.figure(figsize=(12, 6))  sns.lineplot(x=daily\_counts.index, y=daily\_counts.values, marker='o')  plt.title("Số lượt truy cập theo ngày")  plt.xlabel("Ngày")  plt.ylabel("Số lượt truy cập")  plt.xticks(rotation=45)  plt.grid(True)  plt.tight\_layout()  plt.show() |

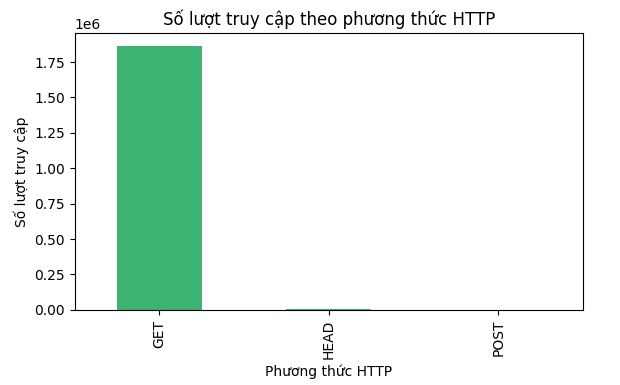
Kết quả:



* + 1. **Biểu đồ số lượng truy cập theo mã phương thức (GET, POST,...).**

|  |
| --- |
| method\_counts = df['method'].value\_counts()  plt.figure(figsize=(6, 4))  method\_counts.plot(kind='bar', color='mediumseagreen')  plt.xlabel('Phương thức HTTP')  plt.ylabel('Số lượt truy cập')  plt.title('Số lượt truy cập theo phương thức HTTP')  plt.tight\_layout()  plt.show() |

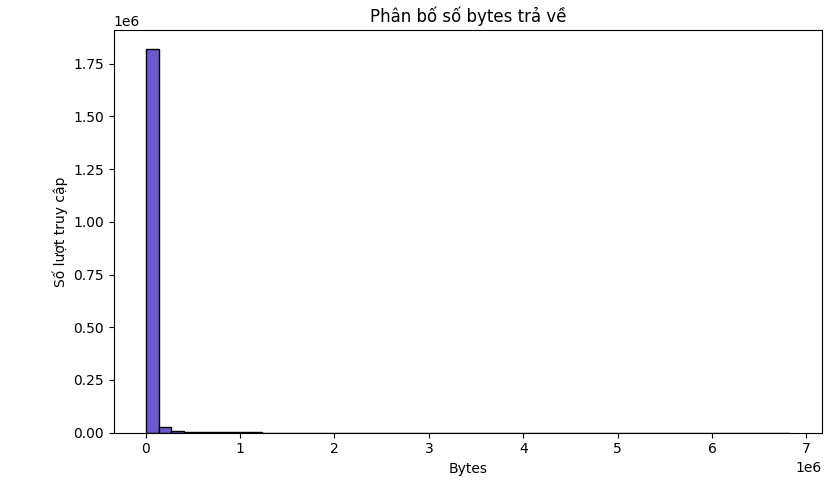
Kết quả:



* + 1. **Biểu đồ phân bố kích thước bytes trả về.**

|  |
| --- |
| plt.figure(figsize=(8, 5))  plt.hist(df['bytes'], bins=50, color='slateblue', edgecolor='black')  plt.xlabel('Bytes')  plt.ylabel('Số lượt truy cập')  plt.title('Phân bố số bytes trả về')  plt.tight\_layout()  plt.show() |

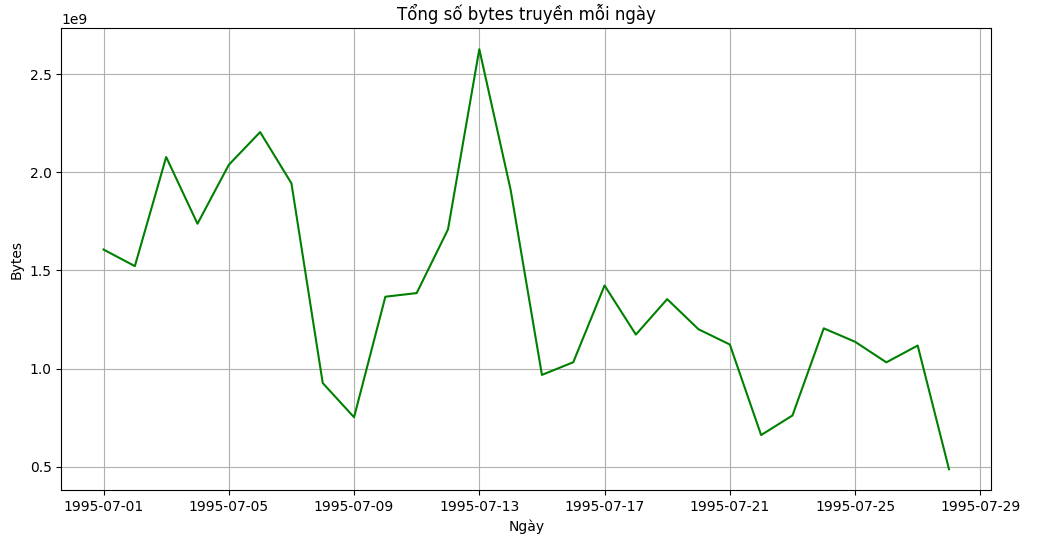
Kết quả:



* + 1. **Tổng số bytes được truyền mỗi ngày.**

|  |
| --- |
| bytes\_per\_day = df.groupby('date')['bytes'].sum()  plt.figure(figsize=(12,6))  bytes\_per\_day.plot(kind='line', color='green')  plt.title("Tổng số bytes truyền mỗi ngày")  plt.xlabel("Ngày")  plt.ylabel("Bytes")  plt.grid(True)  plt.show() |

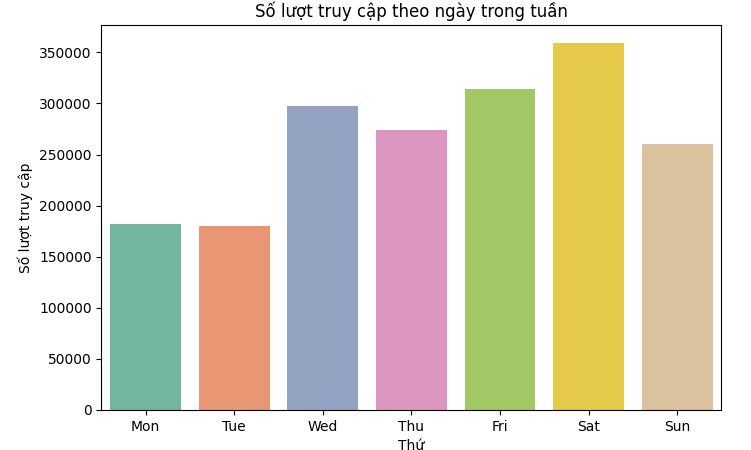
Kết quả:



* + 1. **Số lượt truy cập theo ngày trong tuần (Thứ 2 – Chủ nhật).**

|  |
| --- |
| day\_names = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']  df['day\_name'] = df['ts\_dayOfWeek'].map(lambda x: day\_names[x % 7])  plt.figure(figsize=(8,5))  sns.countplot(data=df, x='day\_name', order=day\_names, palette='Set2')  plt.title("Số lượt truy cập theo ngày trong tuần")  plt.xlabel("Thứ")  plt.ylabel("Số lượt truy cập")  plt.show() |

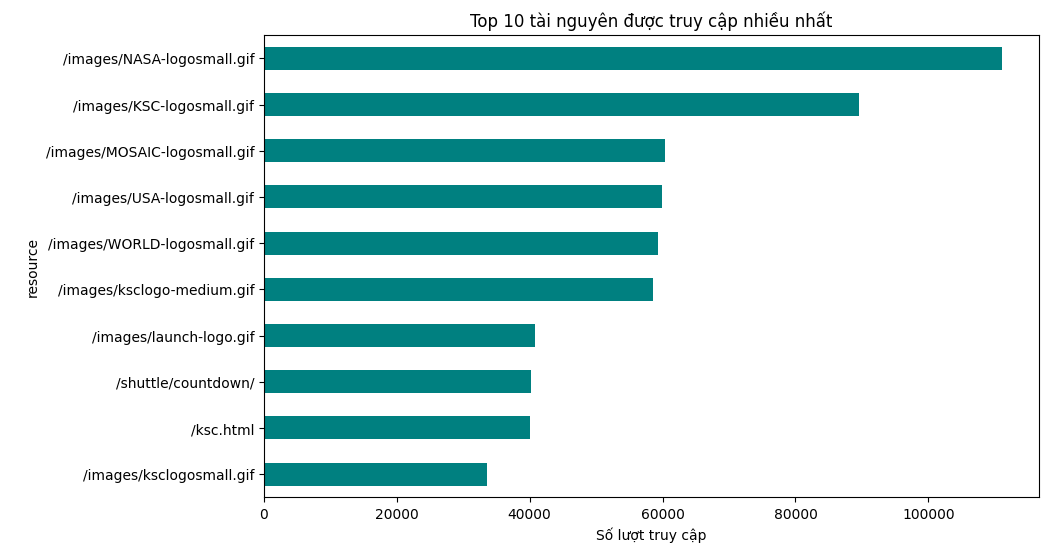
Kết quả:



* + 1. **Top 10 tài nguyên (resource) được truy cập nhiều nhất.**

|  |
| --- |
| top\_resources = df['resource'].value\_counts().head(10)  plt.figure(figsize=(10,6))  top\_resources.plot(kind='barh', color='teal')  plt.title("Top 10 tài nguyên được truy cập nhiều nhất")  plt.xlabel("Số lượt truy cập")  plt.gca().invert\_yaxis()  plt.show() |

Kết quả:



1. **Tổng kết và đánh giá kết quả.**

**5.1 Đánh giá hiệu quả của các bước xử lý và công nghệ sử dụng**

Dự án đã xây dựng thành công một pipeline xử lý dữ liệu log máy chủ web từ việc thu thập, xử lý, phân tích cho đến trực quan hóa. Việc sử dụng các công nghệ thuộc hệ sinh thái Big Data như Apache Flume, HDFS, Apache Spark, Hive và Hadoop Streaming đã chứng minh hiệu quả rõ rệt:

Apache Flume hoạt động ổn định trong việc thu thập dữ liệu log theo thời gian thực và lưu trữ lên HDFS, đảm bảo tính mở rộng và khả năng chịu lỗi cao.

Apache Spark giúp làm sạch và chuyển đổi dữ liệu nhanh chóng nhờ khả năng xử lý song song trong bộ nhớ, đặc biệt hữu ích khi làm việc với hơn 1.5 triệu bản ghi log từ NASA.

Hive cho phép thực hiện các truy vấn tương tự SQL trên tập dữ liệu lớn, hỗ trợ rút trích thông tin một cách dễ dàng và hiệu quả.

Hadoop Streaming kết hợp với các script Python cho phép thực hiện các tác vụ map-reduce linh hoạt như đếm lượt truy cập, phân tích mã phản hồi HTTP, tổng băng thông,...

Các bước trực quan hóa dữ liệu bằng thư viện Python (pandas, seaborn, matplotlib) đã góp phần nâng cao khả năng phân tích, hỗ trợ quá trình trình bày và ra quyết định.

Hệ thống được kiểm thử với tập dữ liệu thực tế, quy mô lớn, và cho kết quả chính xác, hiệu suất tốt, minh chứng cho tính ứng dụng cao của pipeline.

**5.2 Kết luận, bài học kinh nghiệm và đề xuất hướng phát triển**

**Kết luận:**

Dự án cho thấy việc kết hợp các công nghệ Big Data là hoàn toàn khả thi và hiệu quả trong việc xử lý, phân tích và khai thác thông tin từ log máy chủ web.

Qua phân tích, nhóm đã rút ra được nhiều thông tin giá trị như: thời điểm truy cập cao, tài nguyên được yêu cầu nhiều nhất, phân bố mã phản hồi HTTP, và mức độ tiêu thụ băng thông.

**Bài học kinh nghiệm:**

Cần đảm bảo dữ liệu được chuẩn hóa và làm sạch kỹ lưỡng trước khi đưa vào phân tích.

Cần kiểm tra định dạng đầu vào kỹ càng khi tích hợp Flume với HDFS để tránh mất dữ liệu hoặc ghi lỗi.

Trực quan hóa giúp dễ dàng nhìn ra các xu hướng bất thường, nhưng cần lựa chọn biểu đồ phù hợp để tránh gây hiểu nhầm.

Việc sử dụng song song nhiều công nghệ đòi hỏi sự phối hợp và cấu hình kỹ lưỡng, nhất là trong môi trường máy ảo hạn chế tài nguyên.

**Đề xuất phát triển trong tương lai:**

Mở rộng tập dữ liệu sang các tháng khác để phân tích xu hướng theo mùa.

Tích hợp dữ liệu từ nhiều nguồn log khác nhau (access log, error log, security log) để có cái nhìn toàn diện hơn.

Triển khai hệ thống trên cụm Hadoop multi-node, nhằm nâng cao hiệu năng và đáp ứng các yêu cầu ở môi trường thực tế.

Tích hợp mô hình học máy (machine learning) để dự đoán lưu lượng truy cập bất thường hoặc phát hiện tấn công.

Xây dựng dashboard trực tuyến thời gian thực (real-time monitoring) với Apache Kafka và Grafana hoặc Superset.