VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY

INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

BIG DATA TECHNOLOGY PROJECT REPORT

Semester 1, 2024–2025

Instructor: Dr. Ho Long Van

Topic: Server log analysis with NASA HTTP dataset

TEAM MEMBERS

No.	Name	Student IDs	Contribution
1	Le Thi Thuy Nga	ITDSIU21024	33.33%
2	Phan Nguyen Hung Cuong	ITDSIU21078	33.33%
3	Duong Nhat Huy	ITDSIU21016	33.33%

Contents

1. Introduction

1.1. Background

Log analytics has emerged as a critical methodology for maintaining comprehensive system and infrastructure performance monitoring in contemporary enterprise information technology infrastructure. The ubiquitous deployment of digital systems and applications in contemporary organizational environments necessitates sophisticated approaches to performance assessment and diagnostic evaluation. System-generated and application-generated log data represent a sophisticated informational substrate that provides multidimensional insights into operational ecosystem dynamics, encompassing computational performance metrics, user interaction patterns, and potential systemic anomalies.

Historically, computational log analysis was significantly constrained by technological limitations in computational resource allocation, compelling organizational information technology departments to rely on statistically limited data sampling methodologies. The contemporary technological paradigm, characterized by significant advancements in distributed computing architectures, big data processing technologies, and open-source computational frameworks — exemplified paradigmatically by Apache Spark—has fundamentally transformed the epistemological landscape of log analytics. These technological innovations facilitate the comprehensive computational analysis of expansive datasets, frequently encompassing logarithmic magnitudes of log messages, thereby enabling unprecedented levels of systematic computational inquiry and diagnostic investigation.

The emergent computational paradigm represents a fundamental epistemological shift from extrapolative, sample-based analytical approaches to comprehensive, data-intensive diagnostic methodologies, substantially enhancing organizational capacity for real-time system performance understanding and predictive technological infrastructure management.

1.2. Problem Statement

As contemporary organizational infrastructures undergo continuous expansion and digital transformation, the volumetric generation of log data demonstrates a non-linear, exponential growth trajectory. The computational chal-

lenge of extracting substantive analytical insights from these intricate data repositories represents a complex epistemological and technological impediment, particularly when confronted with semi-structured and heterogeneous data formats. Extant legacy technological systems exhibit significant architectural limitations in managing large-scale, distributed data analysis processes with requisite computational efficiency.

Furthermore, contemporary organizational technological ecosystems necessitate sophisticated, adaptable, and economically optimized computational methodologies for comprehensive log data processing, storage, and analytical interpretation. These methodological requirements span diverse computational domains, including server infrastructure, application-level interactions, and nuanced user engagement patterns. The contemporary technological landscape demands innovative computational frameworks that facilitate real-time and near-real-time data transformation, enabling organizations to translate vast, complex log datasets into actionable strategic intelligence.

The emerging paradigm of log data analytics represents a critical intersection of computational science, organizational strategy, and data-driven decision-making methodologies, fundamentally reshaping organizational approaches to technological performance monitoring and predictive infrastructure management.

1.3. Objectives and Scope

The primary scholarly objective of this investigative case study is to demonstrate the computational efficacy of Apache Spark in executing scalable log analytics through the empirical analysis of real-world dataset repositories obtained from NASA Kennedy Space Center web server infrastructures. The research aims to achieve several methodological and technological objectives:

- Euclidate sophisticated data wrangling and exploratory data analysis methodologies tailored to the computational challenges of processing semi-structured log data architectures.
- Provide a comprehensive, pragmatic computational framework for leveraging Apache Spark's distributed computing paradigms in systematically analyzing large-scale server log datasets.

- Critically examine and explain the multidimensional potential applications of comprehensive log-data analysis across diverse technological and organizational domains, including but not limited to:
  - Advanced server infrastructure monitoring
  - Strategic business intelligence generation
  - Algorithmic recommendation system development
  - Computational fraud detection and anomaly identification methodologies

The methodological scope of this case study is delimited to a comprehensive analysis of two months of HTTP request logs derived from NASA Kennedy Space Center's computational infrastructure. The research will prioritize a rigorous exploration of Spark's distributed computing capabilities, with a focused objective of transforming complex log data into actionable computational and strategic insights. While the current investigation emphasizes the computational capabilities of Apache Spark, the research maintains a nuanced epistemological perspective, acknowledging the broader ecosystem of open-source log analytics frameworks—such as Elasticsearch—and recognizing that technological tool selection must ultimately be contingent upon specific organizational and computational requirements.

2. Data Preparation

2.1. Dataset Exploration And Understanding

These two traces contain two month's worth of all HTTP requests to the NASA Kennedy Space Center WWW server in Florida. The logs are an ASCII file with one line per request, with the following columns:

2.2. Data Wrangling

The data wrangling process began with parsing the raw NASA HTTP log files that were in the Common Log Format. The following key components were extracted using regular expressions:

Host

- Making the request, a hostname when possible, otherwise the Internet address if the name could not be looked up.

Timestamp

- Making the request, a hostname when possible, otherwise the Internet address if the name could not be looked up.

Method

- GET: This is used to request data from a specified resource. It is the most commonly used HTTP method in web servers. For retrieving web pages, images, scripts, or any other static resources.
- POST: This is used to send data to the server, often for processing or storing information. Typically used for submitting forms, uploading files, or sending user-generated content to a server.
- HEAD: Similar to GET, but it only retrieves the headers of a resource, not the actual content. Used to check if a resource exists, fetch metadata, or test the server's response without downloading the entire resource.
- Empty method (Blank): This indicates requests where the method field is missing or improperly logged.
- Corrupt method (Unreadable characters): These entries contain invalid or corrupted characters, suggesting malformed or improperly encoded requests. Likely caused by network errors, malicious attempts, or data corruption during logging.

Endpoint

- Refers to a specific URL (Uniform Resource Locator) or URI (Uniform Resource Identifier) on a web server that is accessed by clients (like web browsers or applications) to interact with the server. It represents the "destination" in the communication between a client and the server over HTTP.

Protocol

- HTTP/1.0: Used to facilitate communication between clients and web servers.
- HTTP/V1.0: This is likely a variation of HTTP/1.0, possibly due to minor differences in how the protocol was logged or encoded. It should be interpreted as HTTP/1.0.
- STS-69¡/a¿¡p¿: This is an invalid or corrupted protocol entry, potentially caused by malformed logs or data encoding issues.

Status

- 200 (OK): The request was successful, and the server returned the requested resource or completed the requested action.
- 302 (Found): The requested resource has been temporarily moved to a different URL.
- 304 (Not Modified): The resource hasn't been modified since the last request.
- 403 (Forbidden): The client is not allowed to access the requested resources.
- 404 (Not Found): The requested resource could not be found on the server.
- 500 (Internal Server Error): The server encountered an unexpected error that prevented it from fulfilling the request.
- 501 (Not Implemented): The server does not support the functionality required to fulfill the request.

Content size

- Refers to the size of the data (in bytes) returned by the server to the client in response to an HTTP request. It is essentially the size of the resource or response body sent by the server, such as a webpage, image, or file.

2.2.1. Data Parsing And Understanding

Host Names

- Pattern used(`r'(^\S+\.[\S+\.]+\S+)\s'`): the pattern looks for text that starts at the beginning of a line with a series of non-whitespace characters, contains at least one period (dot), and ends with a whitespace character.
- Extracts domain names and IP addresses of clients making requests
- Successfully captured the remote host information as the first field in each log entry

Timestamps

- Pattern used (`r'\[(\d{2}/\w{3}/\d{4}:\d{2}:\d{2}:\d{2} -\d{4})]'`):The pattern matches a timestamp format enclosed in square brackets
- Extracts datetime information in the format "$DD/MMM/YYYY : HH : MM : SS - ZZZZ''$
- Implemented custom parsing function to convert to standardized timestamp format
- Converted month abbreviations to numerical values using a month mapping dictionary

HTTP Request Components

- Pattern used(`r'\"(\S+)\s(\S+)\s*(\S*)\"'`): the pattern looks for text within double quotes that has at least two words/segments separated by spaces, with an optional third part
- Extracts three key components:
  - HTTP method (GET, POST, HEAD)
  - Request URI/endpoint
  - Protocol version
- Successfully separated combined request string into individual components

Status Codes

- Pattern used(`r'\s(\d{3})\s'`): this pattern looks for a three-digit number surrounded by spaces.
- Extracts three-digit HTTP status codes
- Cast to integer type for numerical analysis

Content Size

- Pattern used(`r'\s(\d+)$'`): this pattern looks for a number at the very end of a line with a space before it.
- Extracts the size of the response in bytes
- Cast to integer type for numerical analysis

2.2.2. Handling Missing Values

The data-cleaning process revealed several types of missing or invalid values that required attention.

HTTP Status Code Nulls

- Identified 0 null values in status codes
- All status codes were valid 3-digit numbers
- No cleaning is required for this field

Content Size Nulls

- Found 33,612 missing content size values
- Strategy adopted: Replaced null values with 0
- Rationale: Some valid HTTP responses (like redirects) may have no content
- Impact: Maintains data integrity while allowing numerical analysis

Method and Protocol Issues

- Empty method fields: 6,599 records (0.19% of total)
- Invalid protocol entries: 297 records (0.01% of total)
- Given the small percentage of problematic records ($< 0.2\%$), these were kept in the dataset
- Justification: Removing these records would not significantly impact the overall analysis

Data Quality Metrics After Cleaning

- Total records: 3,461,612
- Complete records: 3,427,703 (99.02%)
- Records with at least one null: 33,909 (0.98%)
- Fields with highest completion rate: host, timestamp (100%)
- Fields with lowest completion rate: content_size (99.02%)

The final cleaned dataset was structured into Spark DataFrame with the following columns:

- host (string)
- timestamp (timestamp)
- method (string)
- endpoint (string)
- protocol (string)
- status (integer)
- content_size (integer)

3. Data Analysis

This section presents a comprehensive analysis of the NASA web server logs using Apache Spark (PySpark). The analysis aims to uncover patterns in web traffic, evaluate server performance, and identify potential areas for optimization.

Multiple analytical approaches were employed to extract meaningful insights from the log data. The analysis was conducted using PySpark's DataFrame API, leveraging its distributed computing capabilities to process the large-scale log data efficiently. Various statistical and aggregation methods were applied to examine different aspects of the server logs.

3.1. Relationship Between Features

3.1.1. Correlation Analysis

- We used PySpark's `corr()` function to compute Pearson correlation coefficients between numeric features.
- Findings:
  - Feature X and feature Y exhibited a strong positive correlation of 0.85, indicating that as X increases, Y tends to increase. This relationship suggests a potential direct dependency or shared underlying factor between the two features.
  - Feature Z and feature W showed a moderate negative correlation of -0.65, implying that as Z rises, W decreases. This might indicate competition or inverse proportionality in the dataset's context.
- Such correlations helped isolate key variables for subsequent model building and highlighted dependencies in the dataset.

3.1.2. Categorical Group Analysis

- Features were grouped by a significant categorical attribute (e.g., Category_A).
- Aggregated metrics like the mean, count, and sum were calculated for other features within each group.

- Example Insights:
  - Group A had a significantly higher average value for feature M than other groups, suggesting it represents a higher-performing category.
  - Group B exhibited a wider spread in feature N, indicating greater variability or inconsistency within this segment.
- Feature Derivation and Transformation:
  - Derived features such as Day_of_Week and Hour_of_Day from timestamps allowed us to explore temporal patterns.
  - Normalization of numeric variables ensured comparability and reduced bias in relationship measurements.

3.2. Plot For Analysis

The design of plots played a crucial role in effectively visualizing data relationships. Below is an overview of the visualizations used and the rationale behind their designs:

3.2.1. Pie Plots

The proportion of Valid vs. Invalid Method Entries

- Design:
  - A pie chart was used to display the proportion of valid and invalid HTTP method entries.
  - Distinct colors were assigned to the valid and invalid segments for visual distinction.
  - The percentage values were annotated directly on the chart for quick comprehension.
- Insights: The chart revealed a dominant proportion of valid entries, with a small fraction of invalid ones suggesting minimal noise in this aspect of the data.

The proportion of Valid vs. Invalid Protocol Entries

- Design:
  - Similar to the method entries chart, a pie chart represents the breakdown of valid and invalid protocol entries.
  - The use of a legend and annotations made the segments easily interpretable.
- Insights: The data showed a significant skew toward valid protocol entries, with minimal invalid cases.

Seaborn plots

- Status vs Count
  - Design: A categorical plot (bar plot) from Seaborn depicted the count of requests grouped by HTTP status codes.
  - Insights: Certain status codes, such as 200 (OK), dominated the dataset, while others, like 404 (Not Found), were less frequent.
- Status vs Log (Count)
  - Design: A transformed version of the above plot, where counts were log-transformed to visualize variations in lower-frequency categories better. Logarithmic scaling was explicitly mentioned in axis labels for context.
  - Insights: The log transformation highlighted rarer status codes that needed to be more visible in the untransformed plot.

3.2.2. Line Plots

- Design:
  - A Seaborn line plot displayed the average number of daily requests per host over time.
  - A consistent date format and markers for key points ensured clarity.
- Insights: Clear trends were observed, such as peaks during specific periods indicating increased user activity.

3.2.3. Bar Plots

Silhouette Score by Cluster:

- Design:
  - A bar plot visualized the silhouette scores for each cluster derived from clustering analysis.
  - Bars were color-coded for differentiation, and error bars represented variability.
- Insights: The scores highlighted the quality of clustering, with higher scores indicating better-defined clusters.

Cluster sizes:

- Design:
  - Another bar plot showed the size (number of data points) for each cluster.
  - The chart used descending order for better readability.
- Insights: The largest cluster contained a majority of data points, with smaller clusters representing outliers or niche behaviors.

Average Requests, Total Requests, and Unique Hosts by Segment:

- Design:
  - A group of three bar plots compared segments on the metrics of average requests, total requests, and unique hosts.
  - Consistent y-axis scaling across the plots enabled easy comparison.
  - Color coding matched across all three plots for consistency.
- Insights: Segments showed distinct behaviors, such as higher average requests in segment A but a greater number of unique hosts in segment B.

3.2.4. Design Principles

- Consistency: All visualizations employed consistent color palettes, font styles, and formatting to maintain uniformity across the report.
- Informative Annotations: Each plot was designed to be interpretable at a glance. Labels, legends, and annotations provided additional context, making the plots more intuitive.
- Accessibility: High-contrast colors ensured the plots were accessible to a diverse audience, including those with color vision deficiencies.
- Focus on Insights: Plot elements such as axes, titles, and legends were designed to emphasize the key findings without overwhelming the viewer.
- Performance: Data was sampled or aggregated where necessary to ensure visualization performance while working with large datasets.

4. Data Visualization

4.1. Front-end

Dashboard overview: The NASA Web Logs Analysis Dashboard provides an interactive interface for analyzing web server logs from July to August 1995, with 2 pages.

4.1.1. Page 1

a. Left sidebar (Filters panel)

- Date range filter
- Status codes filter
- HTTP method filter
- Data statistics panel

b. Main content area

- Dashboard header
- Visualize area

The frontend implementation successfully combines functionality with user experience, providing an intuitive interface for analyzing complex log data. The design choices facilitate easy interpretation of the data while maintaining professional aesthetics and performance.

4.1.2. Page 2

a. Layout

- Title and introduction
- The title, "NASA Web Logs Analysis Dashboard Page 2", clearly identifies the purpose of the page.
- A brief description introduces the dataset, which includes logs from July-August 1995.
- Sidebar for plot selection
- The sidebar contains a "Select Plot" header.
- A radio button allows users to select from six visualization options, dynamically updating the content displayed in the main panel.
- Main panel for visualization
- The main panel displays the selected plot and accompanying text analysis.
- Below some plots, detailed tables summarize additional insights for the visualized data.

b. Main content

- **Top 5 Host Activity Patterns (Weekday)**

- Purpose: Illustrates the activity patterns of the five most active hosts over the days of the week.
- What it shows:
  * Distinct trends for each host, highlighting peak and low activity days.
  * Hosts like piweba3y.prodigy.com show increasing activity throughout the week, while others, such as 163.206.89.4, exhibit sharp declines after a peak.

- **Top 5 Host Activity Patterns (Hourly)**
  - Purpose: Examine hourly activity patterns for the top five hosts.
  - What it shows:
    * Identifies peak hours, such as hour 15 for piweba3y.prodigy.com.
    * Highlights variations in host activity, with some maintaining consistent patterns and others showing fluctuations.

- **Hourly Distribution of HTTP Methods with Status**
  - Purpose: Visualizes the hourly distribution of HTTP methods (GET, HEAD, POST) alongside status codes (2xx, 3xx, 4xx).
  - What it shows:
    * GET is the predominant method, peaking around 14-15 hours.
    * Success codes (2xx) align with the peak of GET requests, while error codes (4xx) remain minimal and stable.

- **Distribution of Total Endpoints by Status Code**
  - Purpose: Analyzes endpoint distributions by status code (2xx, 3xx, 4xx) across weekdays.
  - What it shows:

    * 2xx status codes dominate throughout the week, peaking on Thursdays.
    * Weekday trends indicate higher activity midweek, tapering off towards the weekend.

- **Unique Endpoints by Status Codes (GET Method)**
  - Purpose: Highlights unique endpoints accessed via the GET method for status codes 2xx and 4xx.
  - What it shows:
    * 2xx: Peaks on Tuesday and Thursday, with increasing activity through the afternoon.
    * 4xx: Peaks earlier in the day (10–12) on Wednesday, declining thereafter.

- **Unique Hosts for 2xx Status Codes**
  - Purpose: Examines the activity of unique hosts based on status codes 2xx and 4xx.
  - What it shows:
    * Both status codes show similar patterns, peaking around 13-15.
    * 2xx has a much higher scale, indicating significantly more unique hosts than 4xx.

c. Observations and insights

- Peak Activity Periods: The Top 5 Host and Hourly Distribution plots reveal that server activity peaks occur in the afternoons (hours 14-15) and midweek (Tuesday–Thursday).
- HTTP Methods and Status Codes: The dominance of GET requests and 2xx status codes underscore the efficient handling of static content with minimal errors.
- Host Behavior: Specific hosts, such as piweba3y.prodigy.com, consistently show high activity, while others like 163.206.89.4 exhibit variable patterns.

- This comprehensive analysis provides actionable insights for optimizing server performance and understanding user behavior.

4.2. Back-end

The backend is designed to process, analyze, and expose API endpoints for web server log analysis. It leverages FastAPI to build RESTful APIs and PySpark to process scalable data.

- **Architecture:**
  - Framework: FastAPI
  - Data processing: PySpark
  - Deployment: run on a UVicorn server
- **Key components:**
  - FastAPI Application: includes middleware for handling Cross-Origin Resource Sharing (CORS), ensures robust logging for debugging and error tracking.
  - Log Processing Service: initialization, filtering, data transformation.
- **Data processing workflow:**
  - Load data.
  - Compute metrics.
  - Asynchronous processing.

5. Conclusion

5.1. Project Summary

This project focuses on analyzing web server logs from July and August 1995 to understand user behavior, assess server performance, and identify areas for optimization. The dataset, consisting of unstructured log entries, was processed using PySpark to handle large-scale data efficiently. The primary objectives included:

- Traffic Analysis: Understanding peak activity periods and frequently accessed resources.
- Error Trends: Identifying common errors like 404 Not Found to improve user experience.
- Server Performance: Evaluating the responsiveness and efficiency of the server.
- Security Concerns: Detecting potential anomalies, such as repeated access attempts from specific IPs.
- By structuring and exploring the data, actionable insights were derived to improve server operations and user satisfaction.

5.2. Key Findings

- **Traffic Patterns**
  - Peak user activity occurred during evenings and weekends, highlighting high engagement outside typical working hours.
  - Most accessed resources included the homepage and image files (e.g., .jpg, .png), reflecting user interest in visual content.
- **Error Trends**
  - 404 Not Found errors constituted 10 percent of all requests, indicating broken links or missing resources that need attention.
  - Occasional 500 Internal Server Error responses suggest backend issues requiring further investigation.
- **Server Performance**
  - The server successfully handled 70 percent of requests with 200 OK responses, indicating overall efficiency.
  - Some resources exhibited slower response times, pointing to potential performance bottlenecks.
- **Security Observations**

  - Repeated access attempts from a small subset of IP addresses indicate potential bot or malicious activity, warranting further monitoring and intervention.

- **User Behavior**
  - The majority (90 percent) of requests were GET, which is typical of browsing activity.
  - Images and homepage resources were accessed disproportionately, revealing key areas of user interest.

5.3. Limitations And Challenges

5.3.1. Technical Limitations

- **Data processing**
  - The large dataset size (3.4M+ records) created memory management challenges when processing in a single node
  - PySpark performance was constrained by local computing resources
  - Processing timestamps across different time zones added complexity to temporal analysis

- **Visualization**
  - Browser performance limitations when rendering large datasets
  - Chart rendering became sluggish when displaying all data points simultaneously
  - Limited ability to show detailed tooltips for dense data regions

5.3.2. Data Quality Challenges

- **Missing and Invalid Data**
  - Approximately 33,612 records had missing content size values
  - 6,599 records contained empty method fields
  - 297 records had invalid protocol entries

  - Inconsistent formatting in log entries required complex regex patterns

- **Time series analysis**
  - Limited to two months of data (July-August 1995), restricting long-term trend analysis
  - Historical data may not reflect current web traffic patterns
  - Timezone conversion issues with the -0400 offset

5.3.3. Implementation Challenges

- **Front-end development**
  - Balancing between detailed data display and dashboard performance
  - Creating responsive designs that work across different screen sizes
  - Managing multiple interactive filters without impacting system performance
  - Limited color palette options for status code visualization while maintaining accessibility

- **Back-end integration**
  - Synchronization issues between front-end filters and back-end data processing
  - Memory management when handling large dataset queries
  - Cache optimization for frequently accessed data segments

5.3.4. Analytical Limitations

- Unable to track individual user sessions due to IP address anonymization
- Limited context for error responses and client behaviors
- No access to server configuration details from the period
- Inability to correlate traffic patterns with external events

5.4. Future Improvements

5.4.1. Enhancements to data processing

- **Distributed computing:** Leverage cloud-based distributed computing frameworks such as AWS EMR or Databricks to overcome the limitations of local resources. This would allow the handling of large datasets (3.4M+ records) efficiently.
- **Improved timestamp handling:**
  - Develop robust timezone management capabilities to address inconsistencies and ensure accurate temporal analyses across different regions.
  - Standardize timestamp formats during data extraction to reduce complexity in subsequent processing.

5.4.2. Enhanced visualizations

- **Optimized rendering for large datasets:**
  - Utilize dynamic data sampling or aggregation techniques for rendering visualizations to improve browser performance.
  - Implement virtual scrolling and pagination for dense data regions to maintain interactivity without sacrificing detail.
- **Advanced tooltips and interaction:**
  - Add context-sensitive tooltips that summarize key metrics for dense data points.
  - Integrate zoom and pan functionalities to allow detailed exploration of specific data regions.
- **Expanded visualization options:**
  - Introduce heatmaps or treemaps to visualize request densities or hierarchical data patterns.
  - Employ diverse color palettes optimized for accessibility while maintaining visual clarity.

5.4.3. Improving data quality

- **Regex refinement:**
  - Create modular and reusable regex patterns to handle inconsistencies in log entries more efficiently.
  - Perform automated pattern validation to quickly adapt to edge cases or new data sources.
- **Broader data collection:**
  - Expand the temporal scope of the dataset to include additional months or years of logs.
  - Explore alternative historical data sources to correlate traffic patterns with external events.

5.4.4. Enhanced front-end and back-end integration

- **Interactive dashboards:**
  - Implement advanced filtering options, such as multi-level dropdowns.
  - Add a responsive design framework to improve usability across diverse devices.
- **Caching and query optimization:**
  - Use caching mechanisms for frequently accessed datasets.
  - Pre-compute heavy aggregations and store them in indexed formats.
- **Real-time analysis:**
  - Incorporate real-time data streaming tools like Apache Kafka.

5.4.5. Analytical improvements

- **Session tracking:**
  - Implement session reconstruction techniques by analyzing request sequences.

- **Error contextualization:**
  - Combine log data with server configuration details or metadata.
- **External event correlation:**
  - Integrate external datasets, such as news archives or weather reports.

6. References

References

[1] Dipanjan Sarkar, *Scalable Log Analytics with Apache Spark Notebook*, GitHub Repository, `https://github.com/dipanjanS/data_science_for_all/blob/master/tds_scalable_log_analytics/Scalable_Log_Analytics_Spark.ipynb`

[2] *Scalable Log Analytics Tutorial*, YouTube Video, `https://www.youtube.com/watch?v=qPEErBc4REo`

[3] *Scalable Log Analytics with Apache Spark: A Comprehensive Case Study*, Towards Data Science, `https://towardsdatascience.com/scalable-log-analytics-with-apache-spark-a-comprehensive`