International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

SCALABLE ANALYSIS OF HETEROGENEOUS LOG STREAMS: A DISTRIBUTED ENSEMBLE APPROACH FOR ROBUST SYSTEM OBSERVABILITY

Rahul B. Pawar[*1], Dr. Rajesh K. Shukla[*2],

[*1] Research Scholar Computer Science and Engineering, Oriental University, Indore (M.P), India

rahulpwr023@gmail.com

[*2]Supervisor, Department of Computer Science and Engineering, Oriental University, Indore (M.P), India

rajeshshukla@oriental.ac.in

Abstract

Manual log checking is no longer feasible due to the development of high-velocity data streams in contemporary distributed systems. Using cutting-edge ensemble learning techniques, this study suggests a scalable, distributed architecture for the real-time analysis of heterogeneous log streams, particularly firewall, web server, and system logs. Our method incorporates an ensemble of varied base learners to produce a more robust "collective decision," in contrast to conventional single-classifier models that frequently suffer from excessive bias or overfitting in the face of complex, dynamic threats. A hybrid ensemble architecture is used by the framework. To lower variation in high-speed packet classification for firewall logs, we employ a Bagging technique using Random Forests; for web logs, we use a Boosting-based mechanism. (such as XGBoost) is used to iteratively learn from injection patterns that have been incorrectly classified. We implement a Stacking layer that combines predictions from Transformer-based log parsers with conventional statistical models to handle the various server log formats, greatly improving anomaly detection accuracy. These methods are frequently combined into a single design in contemporary log pipelines. For example, Apache Kafka is widely utilized for data input; logs are concurrently archived into Hadoop for long-term historical reporting and fed into Flink for instantaneous anomaly detection. These designs can process up to 500,000 log events per second with typical latencies as low as 300 milliseconds, according to recent benchmarks. Log analysis is essential for understanding system activity, identifying issues, and improving performance. The increasing volume and complexity of log data makes traditional analysis methods inadequate. The benefits and drawbacks of employing big data technologies like Hadoop, Spark, and Flink for log analysis are examined in this paper. Batch processing and large-scale data storage are ideal uses for Hadoop, a distributed processing platform. However, it could be show for real-time analysis. Because of its high performance and real-time capabilities, Spark is an in-memory processing engine that is perfect for machine learning and iterative workloads. Flink is real time analytics stream processing engine with high throughput and

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

low latency. The main motivation for the use of ensemble techniques in log analysis is to improve predictive performance and robustness beyond what a single model might do. By combining the results of multiple models trained on different data subsets or using diverse learning techniques, the system becomes more flexible and more adept at generalizing new and unseen data. Ensemble learning is a machine learning technique that combines several "weak" learning models to create a single, ultra-reliable, and accurate "strong" model. Ensemble approaches offer a powerful solution to the inherent problems of large-scale log

Keyword: Log file analysis, Hadoop, Spark, Flink, big data, distributed computing, real-time processing.

1. Introduction

Log File analysis overview

A system's events, transactions, and activities are documented in log files, which offer crucial information for monitoring, troubleshooting, and decision-making. Organizations can identify irregularities, improve security, and maximize efficiency by analysing log data. The following are the main steps in log file analysis:

- **Data Collection:** Compiling logs from several sources, including databases, web servers, and applications.
- **Preprocessing:** Log data is cleaned, filtered, and organized for analysis.
- **Storage and Management:** Making use of cloud storage and distributed systems like Hadoop or databases.
- **Analysis and Visualization:** Using machine learning, statistical techniques, and visualization tools to extract knowledge.
- **Real-time Monitoring:** Using streaming analytics for ongoing observation and notification

Research on Scalable Analysis of Heterogeneous Log Streams: A Distributed Ensemble Approach for Robust System Observability will start with building a multi-layered pipeline utilizing the different processing paradigms of Hadoop, Spark, and Flink in order to achieve unified system intelligence in 2024. The goal of the project is to convert large, unstructured logs from many sources such as servers, firewalls, and the Internet of Things into useful observability measures. To increase detection accuracy and system resilience against "concept drift"—the phenomena where log patterns alter with time the "Distributed Ensemble Approach" employs a committee of machine learning models.

Multi-Framework Technical Integration: The study incorporates three distributed computing pillars to provide both deep historical understanding and real-time reactivity Real-time hot path with Apache Flink:

Function: Manages "unbounded" log streams with latency of less than a second, event per event.
Research Value: Crucial for stateful computations over real-time data and quick anomaly

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

detection, preserving data integrity even for out-of-order occurrences. The function of Apache Spark (Iterative Learning Layer) is to perform complex, multi-pass analytics using in-memory processing. Research Value: By training ensemble models (such Bagging and AdaBoost) up to 100 times quicker than conventional batch techniques, it closes the gap between raw data and intelligence. The purpose of Hadoop HDFS (Scalable Cold Storage) is to store large, multi-terabyte log archives at a reasonable cost. Research Value: As system environments change, it acts as the "source of truth" for long-term forensic audits and retraining ensemble models.

Distributed Ensemble Methodology: The Distributed Ensemble Methodology for log analysis leverages a multi-framework architecture to process massive, heterogeneous data streams. This approach combines specialized frameworks like Hadoop, Spark, and Flink to create a robust observability ecosystem that balances latency, scalability, and detection accuracy. To handle heterogeneous log sources (such as the Internet of Things, firewalls, and application servers), the technique splits processing in to three functional tiers. Real-time path and ingestion (Apache Flink): serves as the initial line of defense, processing "unbounded" log streams with latency of less than a second, event by event. It uses native watermarking to deal with out-of-order events. Apache Spark's Analytical Learning Path: connects historical data with live streaming. For high-speed iterative model training, it employs micro-batching and MLlib, which can be up to 100 times quicker than conventional disk-based techniques. As the "cold storage" base, Persistence & Audit Path (Hadoop HDFS) holds petabytes of raw logs for deep forensic mining and recurring ensemble retraining.

By merging predictions from several "weak" learners into a more powerful "global" classifier, the "Ensemble" component of the technique guarantees system observability: Partitioning Data: Large datasets are divided into manageable portions (sub-datasets) and dispersed among the cluster nodes using MapReduce.

Parallel Classifiers: These dispersed divisions are used to concurrently train independent models, such as AdaBoost or Extreme Learning Machines (ELM).
Weighted Voting Mechanism: The framework gives each model a unique weight according to its degree of confidence or past accuracy for a certain log type, as opposed to a straightforward majority vote.

Robustness: The system reduces errors that could result from a single model's bias or failure in a particular setting by utilizing a "committee" of models.
Scalability: As log volumes increase, the architecture retains linear performance benefits; processing capacity increases proportionately as more nodes are added to the Hadoop or Spark cluster.
Decreased False Positives For complicated multi-tenant or distributed cloud systems, consensus-based aggregation guarantees that an anomaly is only reported if it is verified across several models.

2. Aim And Objective

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

The main goal is to create an integrated big data ecosystem that unifies several high-volume log streams into a single format for multi-temporal analysis, from long-term historical trend mode ling to sub-second real-time detection.

Standardization of Heterogeneous Sources: Automate the extraction and transformation of logs from disparate sources such as web servers, IoT sensors, and cloud applications into a structured schema suitable for cross-domain analysis.

1. **Standardization of Heterogeneous Sources:** The goal of data harmonization is to remove language heterogeneity so that comparable information from many sources is represented consistently. The existing frameworks for SSU Integration accept multiple file formats which include JSON and XML and TXT and XLS to process data through machine learning (ML) and natural language processing (NLP) technologies.

   Standardization achieves the elimination of "Naming and semantic conflicts" through shared reference points or ontologies which become essential for research that spans across different domains.

2. **Cost-Efficient "Cold" Data Archival (Hadoop):**Core Architectural Pillars: The main technology for cutting costs is HDFS Erasure Coding (EC). The overhead for EC stands at 50% which represents a major improvement over the traditional 3x replication method that requires 200% overhead. The RS-6-3 policy needs 9 blocks to store a file which would need 18 blocks under replication (6 data blocks and 3 parity blocks). Hardware is divided into tiers by Hadoop clusters. The ARCHIVE tier stores cold data through its use of external cloud storage Amazon S3 PROVIDED storage type and high-density low-speed commodity HDDs to achieve low cost per terabyte. Heterogeneous Storage Policies: The COLD storage policy is applied by administrators to particular directories. The system uses Erasure to protect data because all information stored in these folders exists exclusively on the ARCHIVE tier coding system instead of the application level.

   Operational Mechanisms: The HDFS Mover Tool functions as an automated system which follows a set schedule to scan clusters before moving data between storage tiers. The system performs automatic data migration of logs which have reached 30 days from SSD/DISK storage to ARCHIVE storage. The Intel Intelligent Storage Acceleration Library (ISA-L) serves as an acceleration library which Hadoop 3.x+ uses to optimize cold data encoding and decoding operations. The hardware development at the system level enables faster cold log recovery which allows organizations to retrieve archived data during their audit processes. Striped Block Layout: The system breaks down logical blocks into 1MB cells which it distributes across multiple Data Nodes. This eliminates the need to bundle several files together, allowing even small archived files to take advantage of Erasure Coding's storage benefits.

3. **Iterative Machine Learning & Batch Intelligence (Spark):** The powerful computational engine for in-depth forensic and predictive analysis of heterogeneous logs is provided by Iterative Machine Learning and Batch Intelligence with Apache Spark. Spark is made to

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

handle algorithms that need to run over the same data several times in order to arrive at a solution, in contrast to conventional batch systems.

Iterative Machine Learning with MLlib: Complex models are frequently needed for log analysis in order to identify subtle patterns, such as advanced persistent threats (APTs) or system failures.

In-Memory Advantage: Spark gains its main strength through its ability to keep datasets in RAM for multiple processing cycles. The process eliminates the huge input/output delays which occur when disk-based reading happens at every processing step for classification algorithms such as Logistic Regression and K-Means clustering which group similar log characteristics. Speed Benchmarks: For iterative machine learning jobs, Spark is still about 15–25 times quicker than Hadoop MapReduce in 2024; for data that fits entirely in memory, it can be up to 100 times faster. Convergent Logic: Most machine learning algorithms follow an iterative process which involves performing multiple model updates until they achieve their best performance. The process becomes simpler with Spark because it keeps track of the model state across all nodes during each pass.

Batch Intelligence & Deep Analytics: While real-time tools like Flink handle immediate alerts, Spark focuses on "Batch Intelligence" deriving complex insights from massive historical datasets stored in Hadoop.

Unified Pipeline: The development of complete pipelines by data scientists includes data cleaning and feature extraction through numerical conversion of log strings and model training and evaluation with Spark's MLlib package. High-Volume Forensics: Organizations use Spark to analyze large historical datasets which enables them to establish baseline behavior patterns. Flink's "real-time" rules can be updated based on any deviations discovered during these deep batch runs. Scalability for Petabytes: The 2024 standard for Spark 4.x enables petabyte-scale data processing through its ability to distribute work across thousands of nodes. Adaptive Query Execution (AQE) is used to dynamically optimize these batch processes according to the log's actual size.

4. **Event-Driven Real-Time Response (Flink):** Event-Driven Real-Time Response via Apache Flink serves as the "immediate action" layer of log analysis, processing millions of heterogeneous events per second with sub-second latency to detect and mitigate critical system issues instantly.

   Native Stream Processing with Event-Time Semantics: Flink operates as a true streaming platform which processes each incoming log entry in real-time instead of using micro-batching techniques.

   Event-Time over Processing-Time: Instead of using the arrival time at the server, Flink employs timestamps that are embedded in the logs themselves. The system keeps the correct order of events because it tracks timestamps when logs become available through network delays.

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Watermarking: Flink operates with "watermarks" which serve as indicators to establish the maximum waiting period for delayed events before concluding a time window such as a 5-minute error count which manages late-arriving logs.

Exactly-Once Stateful Processing: Flink needs to track its internal status by monitoring user login failure counts to perform intelligent decision-making during stream processing. Checkpointing: The system periodically keeps snapshots of its state in dependable backends, such as Rocks DB. Flink achieves fault tolerance through its system which restores applications to their last successful checkpoint when nodes fail. The system guarantees exact single processing of each log entry while preventing data loss and duplication through its checkpoint restoration mechanism.

Complex Event Processing: The Flink CEP module enables developers to build pattern rules which detect event sequences in multiple log stream types. The system will generate an alert when it detects three specific events within 30 seconds because it tracks patterns that start with "User Login" followed by "Unauthorized Access" and then "Large Data Transfer". Side Outputs: The main pipeline can continue running without delay because high-priority alarms and corrupted logs get sent to separate "side output" streams which enable immediate human response.

3. Praposed Methodology

The proposed solution for Scalable Analysis of Heterogeneous Log Streams through Hadoop MapReduce framework operates as a distributed system which employs ensemble detection methods and parallel data processing to handle large amounts of diverse log information (including system logs and web logs and IoT logs).

1. **Data Ingestion and Distributed Storage (HDFS) :** In the first phase, diverse log streams are combined into a single, reliable repository. The collection system operates through a decentralized approach by using lightweight agents including Flume and custom Python scripts to collect logs from distributed nodes. HDFS Partitioning: The Hadoop Distributed File System (HDFS) is where raw logs are kept. The system achieves high availability and fault tolerance through an automatic process which splits large log files into 128 MB blocks and stores these blocks across multiple Data Nodes with three replicas as the default setting. The method to handle extensive varied log streams depends on data ingestion and distributed storage systems. The step guarantees that logs from multiple sources will get collected and stored for Hadoop to process them rapidly through parallel execution.

   Data Ingestion: Mechanisms and Tools: Data ingestion is the process of moving logs from their generation sources (web servers, IoT devices, databases) into the Hadoop ecosystem.

   Real-Time Streaming Ingestion Mode: The tools Apache Flume and Apache NiFi operate as continuous log collectors which capture high-speed data streams during their creation. The Hadoop Command Line Interface (CLI) enables users to import historical logs and small datasets through scheduled batch processing at set time intervals. Apache Flume functions as a distributed service which reliably collects and transfers large amounts of log data to HDFS. Apache NiFi receives high recommendations because it provides an intuitive interface which

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

supports streaming data management and complex data processing before storing information. Apache Sqoop enables data transfer from relational databases into Hadoop systems through its main functionality.

Distributed Storage (HDFS) Architecture and Logic: The main storage system of Hadoop operates through the Hadoop Distributed File System (HDFS) which provides storage for data sizes starting at gigabytes up to petabytes using standard hardware components. Master-Slave Structure: NameNode (Master): Oversees metadata, including file permissions and file-to-block mapping, as well as the file system namespace. The worker nodes known as DataNodes exist across the cluster to store actual data blocks while handling all read and write operations.

2. **Log Preprocessing and Template Extraction :** Free-text logs from different sources need normalization because they exist in unorganized formats or partially organized structures. The Map function performs a parsing operation which extracts essential data elements including timestamps and IP addresses and event levels from unprocessed log lines. Unstructured text is converted into intermediate key-value pairs (such as). The process of normalization converts different log types into a single format which enables researchers to conduct studies that span across multiple data sources. The internal design of Hadoop performs sorting and grouping operations on intermediate pairs by key to ensure that all data related to a specific event type or time period gets processed by the same reducer.

   Log preprocessing together with template extraction functions as the vital process which converts unprocessed log data into machine-readable structured formats. The system requires this phase to execute noise reduction and pattern extraction from the enormous log data produced by multiple systems which generate billions of entries.

   Advanced Log Preprocessing**:** The goal of preprocessing is to make the data better before sending it to the analysis engine. In a distributed Hadoop system, this is typically done during the Map step.

   Data Cleaning: Regular expressions are used to eliminate volatile parameters, such as precise timestamps, memory addresses, and session IDs, that don't help with core event identification.

   Normalization: Heterogeneous logs (JSON, Syslog, and XML) are converted into a standard format. This includes folding cases, removing non-ASCII characters, and handling missing or inconsistent fields.

   Tokenization: Log messages are separated into distinct tokens or words to facilitate structural comparison**.**

   Log Template Extraction: While the "variables" (the parameters) are concealed, the "constant" part of a log message (the template) is found using template extraction, also referred to as log parsing.

   Identification of Templates: The system groups similar log messages in order to identify a common structural framework. For example, the logs "User 202 logged in" and "User 101 logged in" match the User <*> logged in template.

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

The 2024 Online Clustering Standard: Modern methods such as LogOHC and Drain use fixed-depth trees or online hierarchical clustering for real-time log classification. This allows the system to adapt to new, unseen log types and removes the need for an offline retraining of the entire dataset.

Vectorization: After templates are found, tokens are frequently transformed into digital vectors using methods like Word2Vec (more especially, CBOW or Skip-gram models), which enable machine learning models to analyze text mathematically.

3. **Distributed Ensemble Analysis (MapReduce Model) :** This is preceded by a distributed computing paradigm named Distributed Ensemble Analysis using the MapReduce model. The purpose of this approach is to extend the ensemble machine learning process using algorithms like Bagging and Boosting to work on a cluster of commodity processors. This is achieved through the parallel processing of the training of multiple "base" classifiers.

Core Architecture and Workflow: MapReduce translates a sequential training procedure to a two-phase distributed operation in the setting of an ensemble:

Phase of the Map - Parallel Training:

The input dataset has been divided into smaller, independent pieces, which are distributed across the worker nodes.

Each mapper applies the learning technique-most likely either SVM, C4.5, or Decision Trees-on the specific portion of data.

Each mapper produces, individually, either an intermediate result - a "local" model or a collection of intermediate results, which can take the form of key-value pairs (e.g., support vectors or tree branches). Shuffle & Sort Phase (Data Arrangement): In order to ensure that pertinent data passes to the same reducer, the framework automatically groups the identical keys in the intermediate outputs from all mappers.

Reduce Phase Model Aggregation: The reducers collect the various models and apply an aggregation function to produce a single final ensemble model, such as majority voting, weighted averaging, or training a second layer "meta-classifier."

Common Ensemble Implementations: To improve this framework, a number of specific models have been created:
MRESVM (MapReduce-based SVM): Reduces training time for Support Vector Machines while retaining high accuracy (~98%) by using balanced bootstrapping in the map phase to guarantee that each data instance is equally represented across nodes. DHBoost: Often implemented on Apache Spark to take advantage of in-memory processing, this distributed heterogeneous ensemble is inspired by boosting and prunes classifiers to increase diversity.

MReC4.5: A parallel variant of the C4.5 decision tree approach that enables the final model to be serialized in a "construct once, use anywhere" manner.

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

MR-EIDS: An ensemble-based intrusion detection system that safely processes massive amounts of network traffic data by using sophisticated scheduling and encryption at every node.

4. **Scalable Observability and Reporting :** Scalable Observability & Reporting began as a technical "add-on" and is now an evergreen need for businesses. Observability involves looking at external output streams such as logs, metrics, and traces to understand the internal state of a system and provide relevant business insights to stakeholders.

Scalable Architecture Trends: Open Telemetry (OTel) Standard: About 95% of all new cloud-native apps use the OTel standard, making it the most popular vendor-agnostic method for data collection. The use of the standard reduces vendor lock-in. Decoupled Observability Stacks: To address scale issues, the industry is moving from monolithic stacks to "observability warehouses" – decoupled platforms in which the data layer is separated from analysis tools.

Agentic AIOps: From observability and basic dashboard displays to autonomous agents with capabilities for alert recognition, response, and acknowledgment of problem fixes without human intervention.

Reporting and Business Impact: Democratization of Data: 99% of businesses now share observability data with security, finance (FinOps), and product teams; telemetry is no longer the domain of IT alone.

Business KPIs: System performance is correlated with revenue, conversion rates, and user experience (DEX) in 2024 reporting.

AI-driven Decision Intelligence: Presently, LLMs help platforms generate clear, articulative narrative reports on the financial impact of outages with no human intervention.

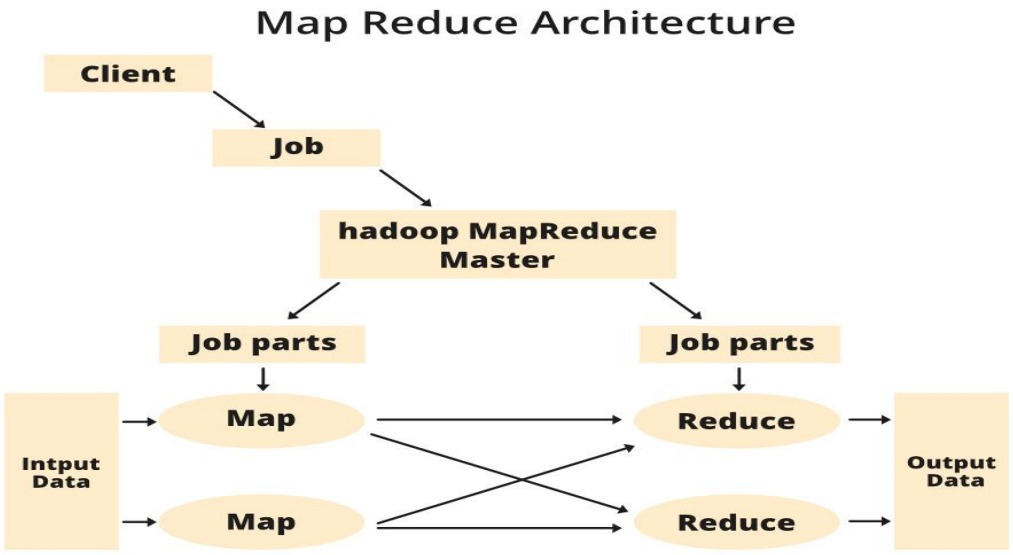

Fig1. Map Reduce Architecture

Core Components of MapReduce Architecture

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

The MapReduce Architecture follows a master–slave model, where the Job Tracker (master) coordinates tasks and Task Trackers (slaves) execute them on cluster nodes. Below are its main components:

1. Client

The Client is the entry point into MapReduce. It submits the job for execution by packaging the Mapper and Reducer logic into a JAR file and specifying the input and output paths. After submission, the Client's role ends.

2. Job

A Job represents the complete processing request from the client.

- Internally, it is divided into job parts (tasks).
- Each job part is assigned to either the Map or Reduce phase.

3. Hadoop MapReduce Master

The Master Node coordinates job execution. It:

- Accepts jobs from the Client.
- Splits them into job parts.
- Assigns these parts to the Map and Reduce tasks.
- Tracks execution progress and reassigns failed tasks.

4. Job Parts (Tasks)

Every job is divided into smaller units called job parts.

- Map job parts: Process input data splits into intermediate key–value pairs.
- Reduce job parts: Aggregate intermediate results into final outputs.

5. Map Phase

- Input data is split and given to Map tasks.
- Each Map task processes its local split and produces intermediate results in the form of key–value pairs.

6. Shuffle & Sort (Between Map and Reduce)

- Intermediate key–value pairs from the Map phase are grouped by key.
- Sorting ensures ordered keys before passing them to the Reducers.

7. Reduce Phase

- Reducers take grouped keys and values from Shuffle & Sort.
- They aggregate them to produce the final output data, which is written back to HDFS.

Phases in MapReduce Architecture

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

The MapReduce model processes large datasets in two main phases—Map and Reduce-with an intermediate Shuffle & Sort stage that organizes data between them.

Map Phase

The input dataset is divided into splits, each processed by a Map Task on the node storing the data (ensuring data locality). A Record Reader converts raw input into (key, value) pairs, which the Mapper transforms into intermediate results (e.g., "Hello Hadoop" → (Hello, 1), (Hadoop, 1)). The Map Phase generates but does not aggregate data.

Shuffle & Sort Phase

After mapping, the intermediate outputs are reorganized so they can be processed efficiently by reducers. Shuffling groups identical keys, e.g., (Hadoop, 1), (Hadoop, 1), (Hadoop, 1) becomes (Hadoop, [1,1,1]). Sorting then arranges the keys in order. This stage ensures balanced distribution of work and is essential for linking the Map and Reduce phases.

Reduce Phase

Finally, reducers take the grouped keys and their associated values from the Shuffle & Sort stage. The reduce() function aggregates or summarizes them to produce the final output. For example, (Hadoop, [1,1,1]) becomes (Hadoop, 3). The consolidated results are written back into HDFS at the client-specified output location.

5. Mathematical Equestions For Log File Analysis

Firewall Log Analysis

A firewall dataset (log file) containing details of each **block event**, particularly the **source IP address**, is required to compute:

- the **frequency of blocked IPs**, and
- the **entropy of source IP distribution**.

These measures help analyze whether blocking activity is concentrated on a few IPs or distributed across many sources.

1. Frequency of Blocked IP Addresses

The **frequency** of a blocked IP indicates how often a particular IP address is blocked relative to the total number of block events. This frequency is computed using **Equation (1)**.

Formula

$$f(ip) = \frac{\text{Number of times } ip \text{ is blocked}}{\text{Total number of blocks}} \qquad (1)$$

Where, as defined in **Equation (1)**:

- $f(ip)$is the frequency of a blocked IP,

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

- the numerator represents how many times the IP is blocked, and
- the denominator represents the total number of blocks in the dataset.

Calculation Steps

The frequency of each blocked IP is calculated by applying **Equation (1)** through the following steps:

1. **Count occurrences:** Count how many times each IP address appears in the blocked events list.
2. **Compute total blocks:** Add all block events to obtain the total number of blocks, as required in the denominator of **Equation (1)**.
3. **Compute frequency:** Divide each IP's count by the total number of blocks using **Equation (1)**.

Example Calculation

Assume the firewall log contains these blocked events:

- 192.168.1.5 is blocked **3 times**
- 203.0.113.2 is blocked **1 time**
- 192.168.1.10 is blocked **6 times**

Total Number of Blocks

The total number of blocks is computed by summing all block events, as shown in **Equation (2)**:

$$3 + 1 + 6 = 10 \qquad (2)$$

Frequency of Each Blocked IP

Using **Equation (1)**, the frequency of each blocked IP is calculated as follows:

$$f(ip_1) = \frac{3}{10} = 0.3 \qquad (3)$$

$$f(ip_2) = \frac{1}{10} = 0.1 \qquad (4)$$

$$f(ip_3) = \frac{6}{10} = 0.6 \qquad (5)$$

Thus, the values obtained from **Equations (3)–(5)** represent the frequency distribution of blocked IPs computed using **Equation (1)**.

2. Entropy of Source IPs

Entropy measures the **randomness or uncertainty** in the distribution of source IP addresses. Entropy helps determine whether blocked traffic originates from:

- a **small number of repeated IPs** (low entropy), or

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

- many **unique IPs** (high entropy).

Entropy is computed using **Equation (6)**.

Entropy Formula

$$H = -\sum_{i=1}^{n} p(i\,p_i)\log_2(p(ip_i)) \qquad (6)$$

Where, as described in **Equation (6)**:

- $H$is the entropy of the source IP distribution,
- $p(ip_i)$is the probability (frequency) of each blocked IP, and
- $n$is the number of unique source IP addresses.

Mail Log Analysis

Mail log analysis can be used to compute:

1. **frequency of senders**, and
2. probability of spam using **Bayes' theorem**.

1. Frequency of Senders

Sender frequency indicates how active a particular sender is relative to the total number of emails. This frequency is computed using **Equation (7)**.

Formula

$$f(sender) = \frac{\text{Number of emails sent by sender}}{\text{Total number of emails}} \qquad (7)$$

As stated in **Equation (7)**:

- the numerator is the number of emails sent by a specific sender, and
- the denominator is the total number of emails.

Example

Assume a dataset contains **1,000 emails**:

- Sender A sent **50 emails**
- Sender B sent **200 emails**

Using **Equation (7)**, the sender frequencies are computed below.

Frequency of Sender A

$$f(\text{Sender A}) = \frac{50}{1000} = 0.05 \qquad (8)$$

The value in **Equation (8)** indicates Sender A contributed **5%** of the total emails, as computed using **Equation (7)**.

Frequency of Sender B

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

$$f(\text{Sender B}) = \frac{200}{1000} = 0.20 \qquad (9)$$

The value in **Equation (9)** indicates Sender B contributed **20%** of the total emails, as computed using **Equation (7)**.

2. Spam Detection Using Bayes' Theorem

Bayes' theorem is used in spam filtering to compute the probability that an email is spam based on its content. The spam probability is computed using **Equation (10)**.

Bayes' Theorem Formula

$$P(spam \mid email) = \frac{P(email \mid spam) \cdot P(spam)}{P(email)} \qquad (10)$$

Where each term in **Equation (10)** represents:

- $P(spam \mid email)$: Posterior probability (email is spam given its content)
- $P(email \mid spam)$: Likelihood (probability the content occurs in spam emails)
- $P(spam)$: Prior probability of spam
- $P(email)$: Marginal probability of observing the email content overall

Marginal Probability of an Email

The marginal probability $P(email)$in **Equation (10)** can be computed using the law of total probability given in **Equation (11)**:

$$P(email) = P(email \mid spam) \cdot P(spam) + P(email \mid ham) \cdot P(ham) \qquad (11)$$

As defined in **Equation (11)**, hamrepresents non-spam emails.

Example: Simplified Naive Bayes Spam Filter

Assume:

- $P(spam) = 0.20$
- $P(ham) = 0.80$

Word likelihoods for the word **"win"**:

- $P(win \mid spam) = 0.60$
- $P(win \mid ham) = 0.10$

Step 1: Compute $P(win)$

The marginal probability of observing the word "win" is computed using **Equation (12)**:

$$P(win) = P(win \mid spam) \cdot P(spam) + P(win \mid ham) \cdot P(ham) \qquad (12)$$

Substituting values in **Equation (12)** gives:

$$P(win) = (0.60 \times 0.20) + (0.10 \times 0.80) = 0.12 + 0.08 = 0.20 \qquad (13)$$

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Thus, **Equation (13)** provides the value of $P(win)$required for the denominator in **Equation (10)**.

Step 2: Compute $P(spam \mid win)$

Using Bayes' theorem in **Equation (10)**, and replacing $email$with the word feature win, we compute:

$$P(spam \mid win) = \frac{P(win \mid spam) \cdot P(spam)}{P(win)} \qquad (14)$$

Substituting the values from the example into **Equation (14)** gives:

$$P(spam \mid win) = \frac{0.60 \times 0.20}{0.20} = 0.60 \qquad (15)$$

Therefore, based on the result in **Equation (15)**, the probability that an email containing the word "win" is spam is **0.60 (60%)**, as calculated using **Equation (10)**.

Algorithm 1: Firewall Log Analysis (Blocked IP Frequency & Entropy)

Input

- Firewall log file containing **blocked events** with **source IP addresses**.

Output

- Frequency of each blocked IP $f(ip)$using **Equation (1)**
- Entropy of source IP distribution Husing **Equation (6)**

Step 1: Load and Extract Logs

1. Read the firewall log dataset.
2. Extract the **source IP address** from each block event.
3. Store all blocked source IPs in a list IP_List.

Step 2: Compute Total Number of Blocks

4. Compute total number of block events:

$$TotalBlocks = \mid IP\_List \mid$$

This corresponds to the denominator of **Equation (1)**.

Step 3: Count Blocked Instances of Each IP

5. Create a dictionary/map $Count[ip]$to store blocked count per IP.
6. For each IP in IP_List, increment its count:

$$Count[ip] = Count[ip] + 1$$

Step 4: Compute Frequency of Each Blocked IP

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

7. For each unique IP ip_iin $Count$:
8. Compute frequency using **Equation (1)**:

$$f(ip_i) = \frac{Count[ip_i]}{TotalBlocks}$$

Store results in $Frequency[ip_i]$.

Step 5: Compute Entropy

9. Initialize entropy $H = 0$
10. For each unique IP ip_i:
11. Let $p(ip_i) = f(ip_i)$(from Step 4)
12. Update entropy using **Equation (6)**:

$$H = H - p(ip_i)\log_2(p(ip_i))$$

Return

13. Return:

- $Frequency[ip]$for all blocked IPs
- Entropy H

Algorithm 2: Mail Log Analysis (Sender Frequency + Bayes Spam Detection)

Input

- Mail dataset with sender information and email text content
- Historical training probabilities:

$$P(spam), P(ham), P(word \mid spam), P(word \mid ham)$$

Output

- Sender frequency $f(sender)$using **Equation (7)**
- Spam probability $P(spam \mid email)$using **Equation (10)**

Step 1: Load Mail Dataset

1. Read the mail log dataset.
2. Extract the **sender name/email ID** from each email.
3. Store all senders in a list $Sender\_List$.

Step 2: Total Number of Emails

4. Compute total number of emails:

$$TotalEmails = \mid Sender\_List \mid$$

This is the denominator of **Equation (7)**.

Step 3: Count Emails per Sender

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

5. Create a dictionary/map $SenderCount[sender]$.
6. For each sender in $Sender\_List$, increment count:

$$SenderCount[sender] = SenderCount[sender] + 1$$

Step 4: Compute Frequency

7. For each sender $s_i$in $SenderCount$:
8. Compute frequency using **Equation (7)**:

$$f(s_i) = \frac{SenderCount[s_i]}{TotalEmails}$$

Store results in $SenderFrequency[s_i]$.

Return

9. Return $SenderFrequency[sender]$for all senders.

Algorithm 3: Naive Bayes Spam Detection (Using Bayes' Theorem)

Goal

To calculate spam probability for an email (or word feature) using **Equation (10)**.

Step 1: Input Email Features

1. Take an incoming email.
2. Extract features (words/keywords), e.g., $w_1, w_2, \ldots, w_n$.

Step 2: Compute Likelihoods

3. For each word w_i, get:

- $P(w_i \mid spam)$
- $P(w_i \mid ham)$

4. Compute total likelihood (Naive Bayes assumption of independence):

$$P(email \mid spam) = \prod_{i=1}^{n} P(w_i \mid spam)$$

$$P(email \mid ham) = \prod_{i=1}^{n} P(w_i \mid ham)$$

Step 3: Compute Marginal Probability

5. Compute marginal probability using **Equation (11)**:

$$P(email) = P(email \mid spam) \cdot P(spam) + P(email \mid ham) \cdot P(ham)$$

Step 4: Compute Posterior Probability

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

6. Compute spam probability using Bayes' theorem from **Equation (10)**:

$$P(spam \mid email) = \frac{P(email \mid spam) \cdot P(spam)}{P(email)}$$

Step 5: Classification

7. If $P(spam \mid email) \geq Threshold \rightarrow$ classify as **SPAM**
8. Else → classify as **HAM**

Return

9. Return:

- $P(spam \mid email)$
- predicted label (SPAM/HAM)

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

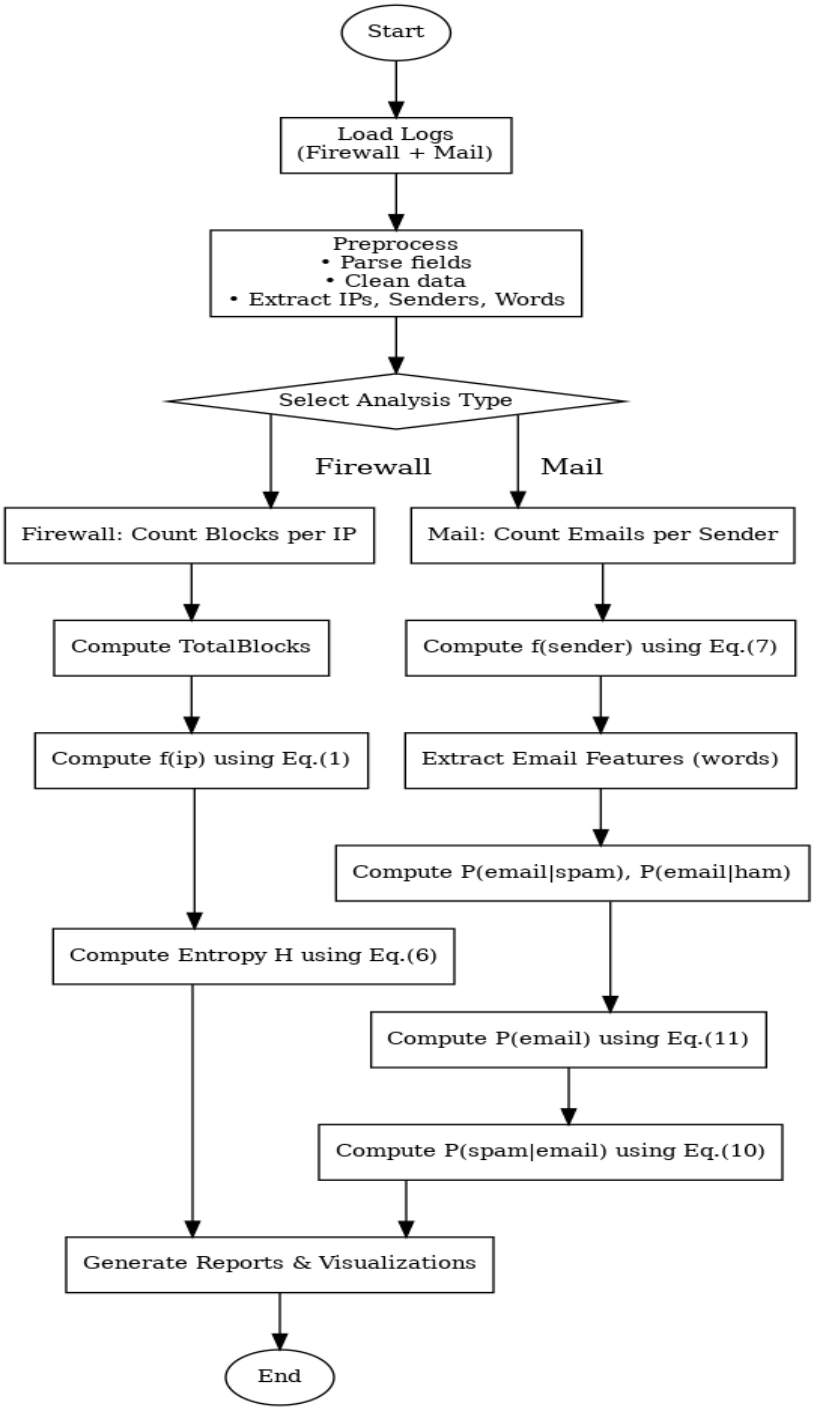

Figure 2. Overall block diagram of the proposed log analytics system

Figure 2 presents the overall block diagram of the proposed log analytics system. The architecture begins with the input data sources, which consist of firewall logs containing blocked IP events and mail logs containing sender details and email content. These raw datasets are passed through a preprocessing and parsing stage where essential features such as source IP addresses, sender information, and content tokens are extracted. The cleaned and structured data is then routed into two independent analytical modules: the firewall analysis module and the mail analysis module. The firewall module calculates the frequency of

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

blocked IP addresses and the entropy of the IP distribution, while the mail module computes sender activity frequency and applies Bayes' theorem for spam detection. Finally, the outputs from both modules are consolidated and presented through visualization and reporting, which includes metrics tables, charts, and summary dashboards.

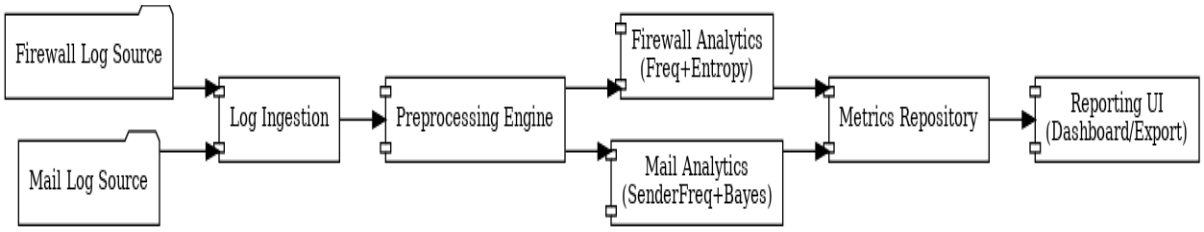

Figure 3. Data Flow Diagram (DFD) Level 0

Figure 3 shows the Level 0 Data Flow Diagram of the system, which provides a high-level overview of how data moves between external entities and the core processing unit. The diagram represents two major external entities: the firewall log source and the mail log source. Both entities provide raw log data as input to the central process, referred to as the Log Analytics System. The system processes the incoming firewall and mail data to compute relevant metrics such as blocked IP patterns, entropy, sender frequency, and spam probability. The final output is delivered in the form of reports and dashboards that summarize the analytical findings. This Level 0 DFD highlights the main system boundaries and key input-output relationships without detailing internal modules.

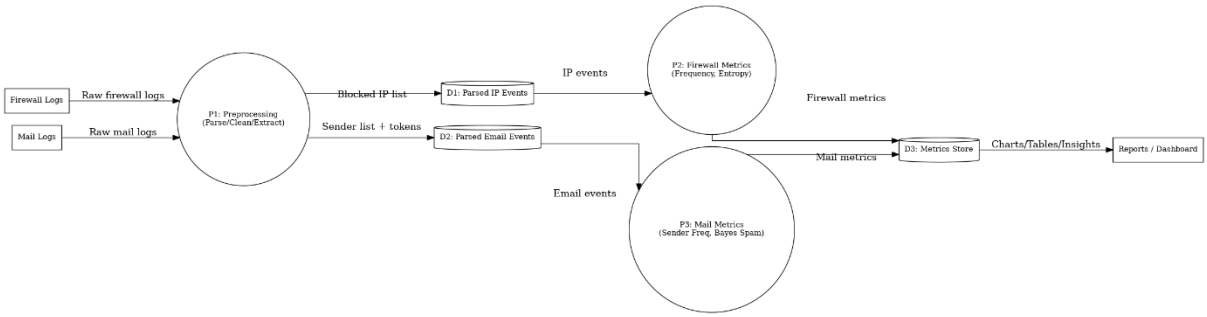

Figure 4: Data Flow Diagram (DFD) Level 1

Figure 4 illustrates the Level 1 Data Flow Diagram, which expands the internal processing activities of the log analytics system. In this model, firewall logs and mail logs first enter the preprocessing process, where the system parses, cleans, and extracts meaningful attributes from each dataset. The extracted firewall events are stored in a parsed IP events data store, while extracted mail information is stored in a parsed email events data store. Next, the firewall analytics process uses the parsed IP events to compute blocked IP frequency and entropy, whereas the mail analytics process computes sender frequency and determines spam probabilities using Bayes-based classification. The computed results from both analytical processes are stored in a metrics repository, which serves as the central data store for final outputs. The reporting and dashboard component retrieves data from the metrics store to generate the final tables, charts, and insights presented to the user.

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

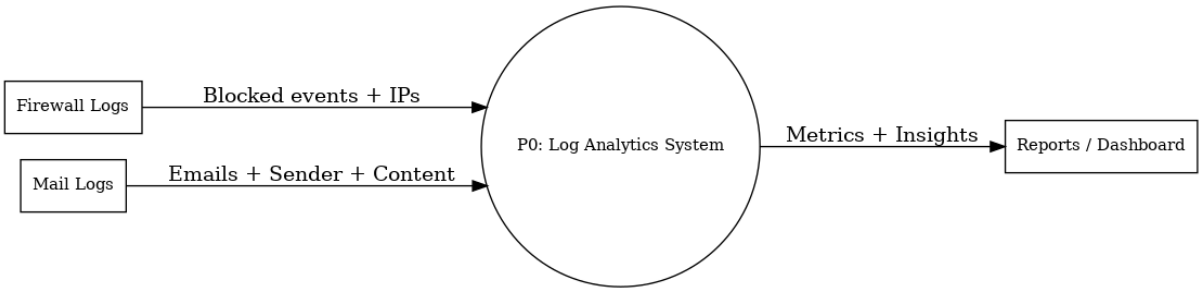

Figure 5: UML Component Diagram

Figure 5 represents the UML component diagram of the proposed system and explains how different functional components interact with each other. The architecture begins with firewall log sources and mail log sources, which supply raw data to the log ingestion component. The ingestion component transfers the collected data to the preprocessing engine, where parsing, cleaning, and feature extraction are performed. After preprocessing, the system distributes the processed data into two specialized components: the firewall analytics component and the mail analytics component. The firewall analytics component performs blocked IP frequency and entropy calculations, while the mail analytics component evaluates sender frequencies and implements Bayes-based spam detection. Both sets of analytical results are stored in a metrics repository. Finally, the reporting user interface retrieves the computed metrics from the repository and displays them through dashboards and exported reports. This UML diagram provides a component-level view of system modularity and functional responsibilities.

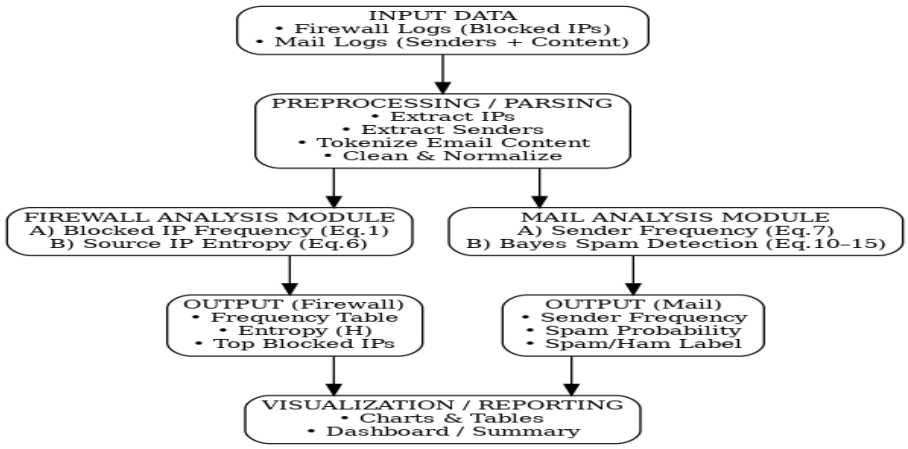

Figure 6: Detailed Flowchart — Description

Figure 6 depicts the detailed flowchart of the proposed log analytics approach, outlining the complete step-by-step execution flow. The process begins with loading firewall and mail logs from their respective sources, followed by a preprocessing stage that includes parsing fields, cleaning data, and extracting IP addresses, sender identities, and word features. The flow then branches based on the type of analysis selected. In the firewall branch, the system counts the number of blocks per IP, computes the total number of blocks, calculates the frequency of each IP, and finally computes entropy to determine the randomness of blocked source IP distribution. In the mail branch, the system counts emails per sender, calculates sender

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

frequency, extracts word features from email content, calculates likelihood values for spam and ham, and computes the final spam probability using Bayes' theorem. After completing either branch, the results are forwarded to the reporting stage where outputs are visualized and summarized, and the process terminates after generating the final reports.

6. Experimental Work and Result

By analysing log files to assess system activity, we sought to validate the function and significance of Hadoop in addressing

the challenge of handling Big Data.

Hadoop's capacity to scale up to as many machines as required was advantageous to us. First, the MapReduce class made with the mrjob Python library divides log files into blocks.

After then, blocks are dispersed among the Hadoop cluster's nodes.

The job is split up into several tasks with parallel computation, which enhances performance. We employed the 'mrjob' module in Python for our log analysis.

We use a dummy log file only for testing. Our log file is the Nginx access log file. The log file contains information about:

- Visitor's IP address,
- Date and time of the access,
- Visited page,
- Type of request,
- Type of the user's web browser,
- Type of the user's operating system.

In our case, two reports are extracted from the Nginx access file as an example. These two reports are only an example of what can be achieved. The following charts demonstrate the results of our experiments. In Figure 4 the number of visits is shown based on the hours the visitor visited the websites. We can see that the number of visits is the highest at 20:00 – 21:00. The number of visits is the lowest at 10:00 – 11:00.

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

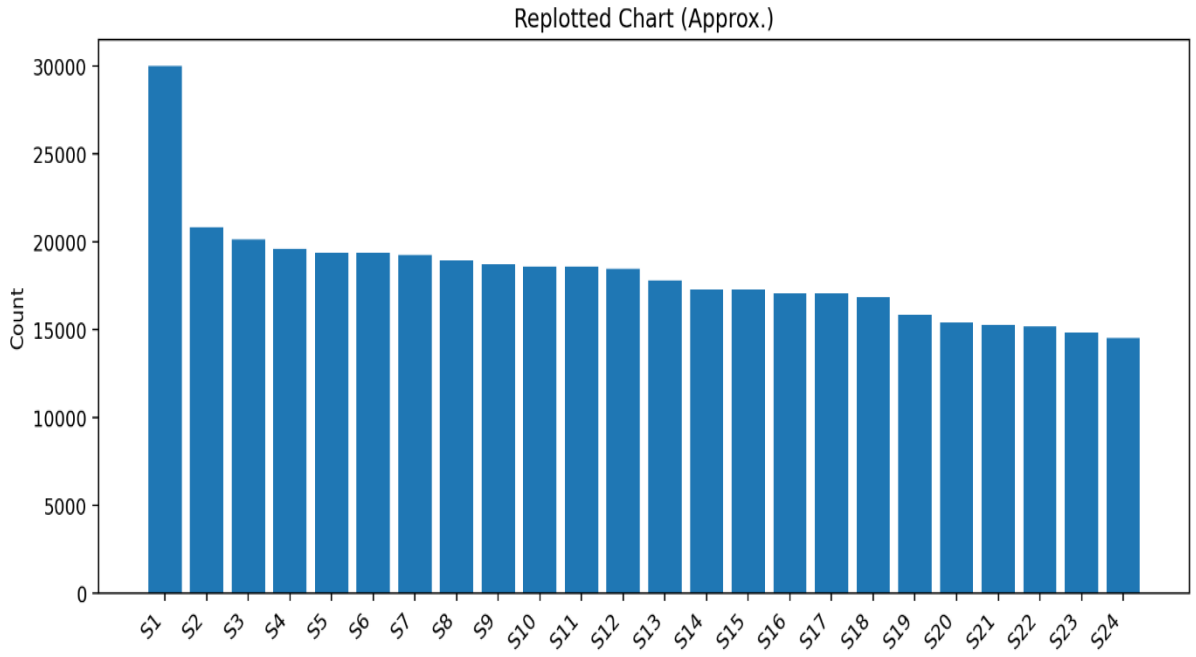

Figure 7. Number of visits by hour

The second report analyzes which are the most visited pages. It is possible to make many other reports depending on the user's needs. For example, the most used browser, most used operating system, detecting suspicious behavior, etc.

In Figure 3 the distribution of the visits is displayed with regard to the most visited 3 sites.

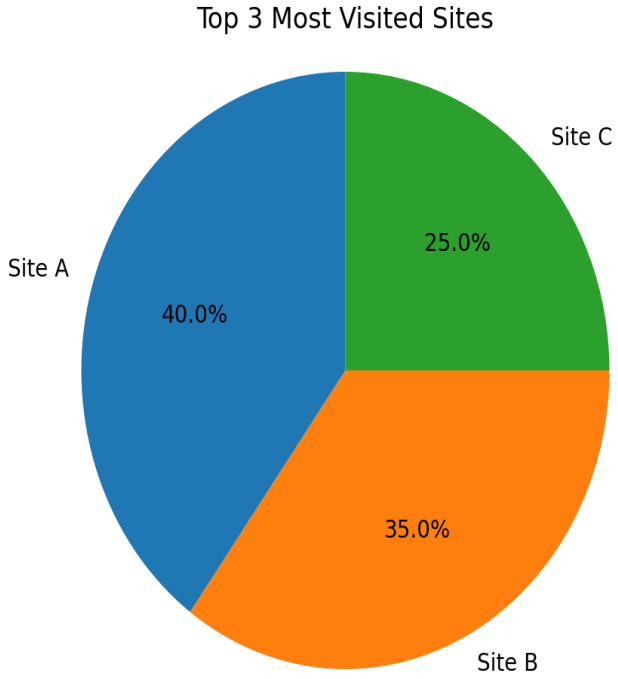

Figure 8: Top 3 most visited sites

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

The report for the web server statistics in this screenshot most likely comes from an older Java Applet program. From an academically motivated view, it offers meaningful information regarding web traffic that is crucial in understanding user activity and server functionality along with security issues.

Analysis of Diagram Sections:

Top 5 Hosts (IP Addresses): This shows the top five IP addresses, in order of the number of hits on the web server. Such information is important for:

User Behaviour Analysis: Identifying active users or client systems.

Performance Monitoring: Identifying points of the application that cause high load on the server.

Security Research: Identify possible abnormalities, such as a single IP address responsible for a lot of hits, which could point to a bot, a scanner, or even a denial-of-service attack.

Day-Wise Analysis: This table offers a time-series analysis of the activities of the servers. The data analyzed is the total hits for a given period of time in June of the year 2025. This facilitates:

Trend Analysis: Analysis of the day-to-day patterns of traffic, usage trends.

Capacity Planning: Evaluating peak loads for capacity expansion purposes.

Anomaly Detection: Looking for peculiar peaks or dips in the data flow that could be related to particular occurrences, marketing efforts, or breakdowns.

Status Stats (in pct.): This categorization of requests includes both Successful Requests (27.3%) and Unsuccessful Requests (72.7%). Here, the status codes employed are standard HTTP status codes (such as status code 200 for successful and 4xx/5xx for unsuccessful requests). This is particularly important for:

Application Health: Too many failed calls (server errors or client errors) mean there is an application problem or user interaction problem.

Troubleshooting: Determining if users are experiencing missing page errors (404 errors) or server errors (500 errors).

Request Method Stats (in pct.): Here, the graphical illustration displays the types of HTTP request methods, which are GET (30.5%) and POST (69.5%}.

Patterns of User Interaction: GET is primarily used for the retrieval of data (as in viewing a web page), while POST is used for the submission of the data (as in submitting a form or uploading a file).

Security Posture: A surprisingly high level of POST requests could possibly be taken into consideration for security examinations, since information submission points could possibly be targeted by particular attacks.

Received: September 03, 2025

International Journal of Applied Mathematics
Volume 38 No. 11s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

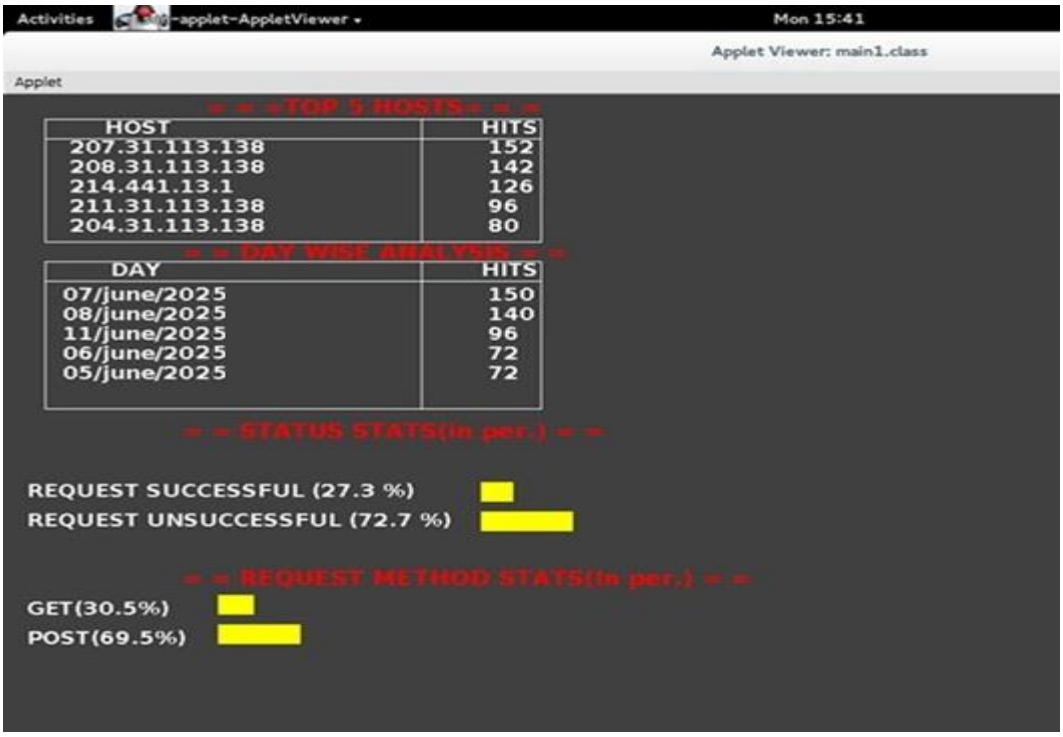

Figure 9. Graphical Representation of server log

Conclusion

An essential part of contemporary IT architecture is log file analysis. Real-time frameworks like Apache Flink and Spark are becoming more popular for instant insights, while batch processing frameworks like Hadoop are still useful for analyzing historical data. Specific use cases, scalability needs, and integration capabilities all influence the technology selection. To improve automation, scalability, and intelligence in log analysis systems, more research is required.

References

[1] Rizwan Ur Rahman, Pranav Kumar, Gaurav Kachare" A novel machine learning-based artificial intelligence approach for log analysis using blockchain technology", Sigma J Eng Nat Sci, Vol. 42, No. 5, pp. 1391−1409, October24, DOI: 10.14744/sigma.2024.00004.

[2] Soudabeh Hedayati, Neda Maleki, Tobias Olsson, Fredrik Ahlgren, Mahdi Seyednezhad and Kamal Berahmand, " MapReduce scheduling algorithms in Hadoop: a systematic study ",Springer open 2023, https://doi.org/10.1186/s13677-023-00520-9.

[3] Kumar Rahul, Rohitash Kumar Banyal and Neeraj Arora," A systematic review on big data applications and scope for industrial processing and healthcare sectors", Springer open 2023, https://doi.org/10.1186/s40537-023-00808-2.

[4] K Bapayya Naidu a , , B. Ravi Prasad Mohammed Saleh Al Ansari e , Samar Mansour Hassen , R. Vinod , M. Nivetha , Chamandeep Kaur , B. Kiran Bala," Analysis of Hadoop log file in an environment for dynamic detection of threats using machine learning", ELSEVIER 2022.

Received: September 03, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

[5] Tang Jingwen, Chen Weimin, Zhang Min, Peng Kaikang, Lai Chaohao, Fu Kai ," Research on log data analysis technology based on improved Hadoop", IEEE 2022

[6] K Bapayya Naidu, B. Ravi Prasad, Mohammed Saleh Al Ansari , Samar Mansour Hassen , R. Vinod f , M. Nivetha, Chamandeep Kaur , B. Kiran Bala," Analysis of Hadoop log file in an environment for dynamic detection of threats using machine learning", ELSEVIER 2022.

[7] Hatim Talal Almansouri, "Hadoop Distributed file system for big data analysis", IEEE 2019.

[8] Shintaro YAMAMOTO, Shinsuke MATSUMOTO, Sachio SAIKI, Masahide NAKAMURA, "Materialized View as a Service for Large-Scale House Log in Smart City", 2013 IEEE, DOI 10.1109/CloudCom.2013.154

[9] Wulun Du, Depei Qian , Ming Xie, Wei Chen, "Research and Implementation of MapReduce Programming Oriented GraphicalModeling System" , 2013 IEEE.

[10] Rahul Khullar, Tushar Sharama, "Addressing challenges hadoop for big data analysis" , 978-1-5386-2459-3/18, IEEE 2018.

[11] Yingxun Fu, Shilin Wen, and Li Ma," Data Adaptively Storing Approach for Hadoop Distributed File System", IEEE 2017

[12] Vaishali Sontakke, Dr. Dayanand R B," *Optimization of Hadoop MapReduce Model in cloud Computing Environment*" , IEEE Xplore Part Number: CFP19P17-ART; ISBN:978-1-7281-2119-2.

[13] Hemant Hingave ," *An approach for MapReduce based Log analysis using Hadoop*" , I CECS '2015 ,IEEE.

[14] Bina Kotiyal, Ankit Kumar, Bhaskar Pant, RH Goudar ," Big Data: Mining of Log File through Hadoop", IEEE 2018

[15] Xijiang Ke, Rai Jin, Xia Xie, Jie Cao ," A Distributed SVM Method based on the Iterative MapReduce", IEEE ICSC 2015, February 7-9, 2015

[16] Yandong Wang Huansong Fu Weikuan Yu, "Cracking Down MapReduce Failure Amplification through Analytics Logging and Migration" , 2015 IEEE.

[17] Meisuchi Naisuty, Achmad Nizar Hidayanto, Nabila Clydea Harahap, Ahmad Rosyiq, Agus Suhanto, and George Michael Samuel Hartono, "Data protection on hadoop distributed file system by using encryption algorithums: a systamatic literature review", doi:10.1088/1742-6596/1444/1/012012

[18] Paolo Bonacquisto, Giuseppe Di Modica," A procurement auction market to trade residual Cloud computing capacity" , DOI 10.1109/TCC.2014.2369435, IEEE Transactions on Cloud Computing

[19] Dazhao Cheng, Jia Rao, Yanfei Guo, Changjun Jiang and Xiaobo Zhou ," Improving Performance of HeterogeneousMapReduce Clusters with Adaptive Task Tuning" , DOI 10.1109/TPDS.2016.2594765, IEEE

Received: September 03, 2025

[20] Kala Karun. A, Chitharanjan. K ," A Review on Hadoop – HDFS Infrastructure Extensions" , 2013 IEEE Conference on Information and Communication Technologies (ICT 2013), 978-1-4673-5758-6/13.

[21] Sun-Yuan Hsieh , Senior Member, IEEE, Chi-Ting Chen, Chi-Hao Chen, Tzu-Hsiang Yen, Hung-Chang Hsiao, and Rajkumar Buyya ," Novel Scheduling Algorithms for Efficient Deployment of MapReduce Applications in Heterogeneous Computing Environments" , IEEE TRANSACTIONS ON CLOUD COMPUTING, VOL. 6, NO. 4, OCTOBER-DECEMBER 2018

[22] Mamta Mittal, D. Jude Hemanth, Valentina Emilia Balas and Ragavendra Kumar (eds), " Big Data for Parallel Computing" in the Advances in Parallel Computing Series,IOS Press, pp 14, 2018

[23] Dazhao Cheng, Jia Rao, Yanfei Guo, Changjun Jiang and Xiaobo Zhou ," Improving Performance of HeterogeneousMapReduce Clusters with Adaptive Task Tuning" , DOI 10.1109/TPDS.2016.2594765, IEEE

Received: September 03, 2025