Adaptive Threshold Determination for Syslog Monitoring by Exponential Moving Average

Muhammad Akram[1] Tomoyuki Koyama[2] Takayuki Kushida[1]

Abstract：Syslog records event messages from computer systems and network devices, and the Cloud and Distributed Systems Laboratory (CDSL) collects approximately 3 to 5 million syslogs per day. "ElastAlert" monitors these logs but uses static thresholds that cannot adapt to changing log patterns. This study proposes a dynamic threshold method using the Exponential Moving Average (EMA) to adjust alert sensitivity based on recent warning log counts. A program retrieves logs from "Elasticsearch", calculates EMA values, and updates the threshold automatically. The basic experiment showed that $n = 3$ and $k = 1.5$ provided stable performance. The evaluation applied the dynamic threshold to the CDSL system, and the alert accuracy for container logs was approximately 46.2%, 36.8%, and 66.7% for the alertmanager-error, prometheus-error, and prometheus-warning rules. All hypervisor and hardware alerts were false, resulting in 0.0% accuracy for the rose-warning, wifi-warning, and NAS-warning rules. The overall accuracy was approximately 24.9%. The results show that the EMA-based threshold improved alert behavior for container logs but requires refinement for syslog-level warnings and filtering accuracy.

1. Introduction

Background

Syslog is a standard protocol that records event notification messages generated by computer systems and network devices [1]. The Cloud and Distributed Systems laboratory (CDSL) is a research laboratory inside Tokyo University of Technology. CDSL students use VMware ESXi every day to conduct research and experiments for thesis work. Servers and network devices produce syslogs. CDSL collects around 3 to 5 million syslogs per day. Some of the syslogs contain contents that require alerting, as shown in Code 1. The warning varies from hardware faults, internal failures and system availability. The severity levels are also included in syslogs and indicates whether the message contains error logs or other important events.

Code 1: Warning syslog example

```
1 <180>2025-09-24T01:32:04.578Z
  mint.a910.tak-cslab.org vmkwarning:
  cpu0:2097334)WARNING: HBX: 2723: Failed to
  cleanup registration key on
  volume643e513a-12218ac1-de51-1cfd08797254:
  Failure
```

Code 1 shows a Warning syslog example. The value inside angle brackets <180> is the PRI part, which represents the priority of the message by combining facility and severity values [2]. The timestamp "2025-09-24T01:32:04.578Z" indicates the exact time the event occurred. The hostname "mint.a910.tak-cslab.org" identifies the source device that generated the log. The process name "vmkwarning" shows the component that produced the message, and the text after the colon describes the event details. The syslog message provides key information for identifying the origin, time, and severity of system events.

Figure 1 shows the Syslog monitoring flow. Logstash collects the syslogs and sends the logs to Elasticsearch. Kibana connects to Elasticsearch and allows

[1] Tokyo University of Technology, Department of Computer Science
1404-1 Katakuracho, Hachioji-shi, Tokyo 192-0982
[2] Tokyo University of Technology, Graduate School of Bio-Information and Media Science, Department of Computer Science
1404-1 Katakuracho, Hachioji-shi, Tokyo 192-0982

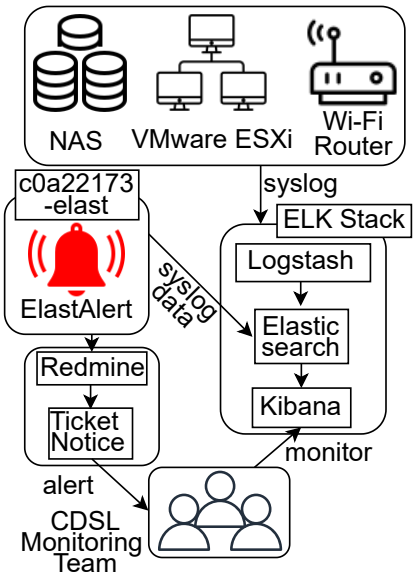

Figure 1: Syslog monitoring flow

CDSL members to monitor the syslogs on dashboards [3, 4]. ElastAlert that is set up on a VM called "c0a22173-elast" checks the syslogs stored in Elasticsearch. ElastAlert detects an anomaly and sends an alert to Redmine. A system called "Ticket Notice" is registered inside Redmine to automatically notify the CDSL monitoring team if a new ticket is registered. The CDSL monitoring team accesses Redmine to confirm the alerts without constant manual monitoring of dashboards.

Main Issue

ElastAlert requires a fixed threshold inside the YAML configuration file. ElastAlert cannot change the threshold according to changes in the warning syslog count.

Code 2: YAML file metrics

```
1 type: frequency
2 num_events: 625
3 timeframe:
4   minutes: 60
5 filter:
6 - term:
7     log.syslog.severity.name.keyword: Warning
8 - term:
9     host.hostname.keyword: lily
```

Code 2 shows YAML file metrics for ElastAlert. Line 1 defines the rule type as frequency, and the frequency type counts how many times a condition occurs within a specific timeframe. Line 2 sets num_events, and num_events controls how many warning logs must occur before ElastAlert sends an alert. Line 3 sets the timeframe field, and the timeframe defines the period used to evaluate the number of warning logs. Line 4 sets the unit of the timeframe, and the unit in this case is minutes. Line 5 starts the filter section, and the filter applies conditions to narrow down the logs. Line 6 adds a filter condition for severity level equal to warning. Line 8 adds a filter condition for hostname equal to lily, which is a VMware ESXi host. Lines 6 to 9 uses an AND condition for the filters.

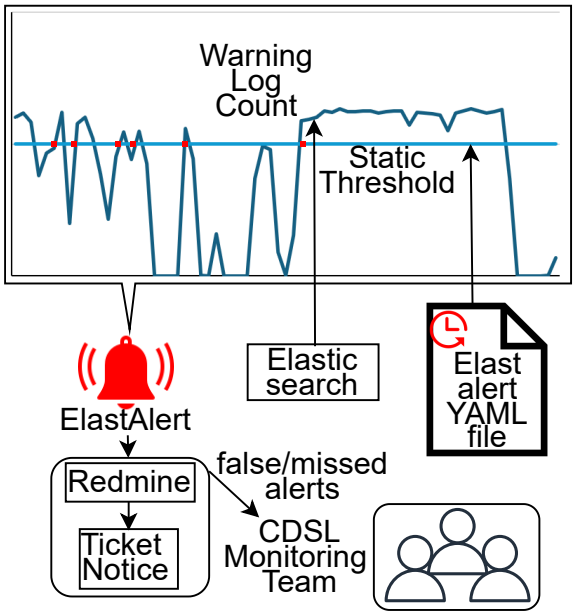

Figure 2: Static thresholds lead to false/missed alerts

Figure 2 shows Static thresholds lead to false/missed alerts. ElastAlert applies several static thresholds as horizontal lines. Warning log counts cross the static thresholds during normal usage and during error bursts. ElastAlert sends alerts when warning log counts cross the thresholds. Static thresholds cause false alerts during benign spikes and cause missed alerts during abnormal activity below high thresholds. CDSL monitoring team receives alerts from ElastAlert through Redmine. False alerts increase workload for CDSL monitoring team and missed alerts delay response to real problems [5]. Dynamic behavior of warning log counts conflicts with static thresholds and reduces alert reliability.

Overview of Each Section

Section 2 describes related studies. Section 3 describes the proposed method and use case scenario. Sec-

tion 4 describes the implementation. Section 5 describes the experimental environment, the proposed method, and an evaluation of the experimental results. Section 6 provides a discussion. Section 7 is a summary of this paper.

2. Related Studies

Previous work has explored dynamic threshold mechanisms in log acquisition systems. One study designed a system that used an autoregressive model to adjust thresholds automatically based on log patterns [6]. The approach improved adaptability compared to static thresholds and maintained reliable log collection performance. This study relates to this research by also using a dynamic threshold mechanism but applies the Exponential Moving Average (EMA) method to adjust alert thresholds for syslog monitoring instead of log acquisition.

Other research applied the Exponential Weighted Moving Average (EWMA) technique together with a predictive model to improve the accuracy of performance metric analysis [7]. The method focused on enhancing trend sensitivity and reducing prediction errors in time-series logs. This study uses the EMA concept in a different way by calculating a dynamic threshold for ElastAlert, allowing real-time adaptation to changes in warning log counts inside Elasticsearch.

Another study has integrated the ELK Stack with external systems to detect cyber attacks and identify network anomalies [8]. The combination of Elasticsearch, Logstash, and Kibana was used to automate event detection and analysis across large datasets. This study applies a similar ELK Stack structure but focuses on detecting performance bottlenecks and operational problems in VMware ESXi by integrating ElastAlert for automatic alerting in infrastructure monitoring.

3. Proposal

This paper proposes a method to decide a dynamic threshold for ElastAlert [9]. The dynamic threshold is set hourly by a program that collects syslogs from Elasticsearch [10]. The hourly interval allows the threshold to adapt to log pattern changes at a suitable and stable rate. The program calculates the Exponential Moving Average (EMA) of warning syslog counts and applies calculations with different variables to generate a threshold [11]. The variables can be altered to adjust sensitivity. The dynamic threshold adapts to changes in warning syslog counts inside the CDSL environment.

EMA Formula

Several components need to be considered when calculating the dynamic threshold for ElastAlert. The following three equations explain the formulas used for the calculation.

$$\mathrm{EMA}_t = \alpha \cdot x_t + (1 - \alpha) \cdot \mathrm{EMA}_{t-1} \tag{1}$$

Equation 1 calculates the Exponential Moving Average (EMA) [12]. x_t is the current warning log count at time t. EMA_{t-1} is the EMA value at the previous time step. α is the smoothing factor that controls the weight of the current log count, calculated in Equation 2. A larger α makes the EMA follow the latest log count more closely, while a smaller α makes the EMA change more slowly. The EMA smooths log fluctuations and provides a stable base for the threshold calculation.

$$\alpha = \frac{2}{n+1} \tag{2}$$

Equation 2 calculates the smoothing factor α from the smoothing period n. The variable n is a configurable value that controls how quickly the EMA reacts to recent warning log changes. A smaller n produces a larger α and the EMA adapts faster. A larger n produces a smaller α and the EMA adapts more slowly. The value of n also defines how much past logs is used in the σ calculation.

$$\mathrm{Threshold} = \mathrm{EMA}_t + k \cdot \sigma \tag{3}$$

Equation 3 calculates the dynamic threshold. EMA_t is the current EMA value. σ is the standard deviation of recent warning log counts, calculated from past logs based on the n value. k is a configurable variable that adjusts the strictness of the threshold. A larger k produces a higher threshold and fewer alerts, while a smaller k produces a lower threshold and more alerts. The formula adapts the threshold to the recent pattern of warning logs. The threshold follows log trends and reduces false alerts in ElastAlert.

Use Case Scenario

Figure 3 shows the Use case scenario for dynamic threshold monitoring in CDSL. Syslogs are collected and visualized inside Kibana. CDSL monitoring team members can manually monitor syslogs on the dashboards to check for abnormal log activity. ema.py calculates the Exponential Moving Average (EMA) and sets

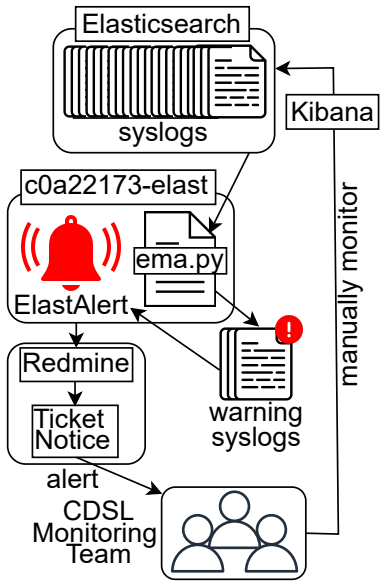

Figure 3: Use case scenario for dynamic threshold monitoring in CDSL

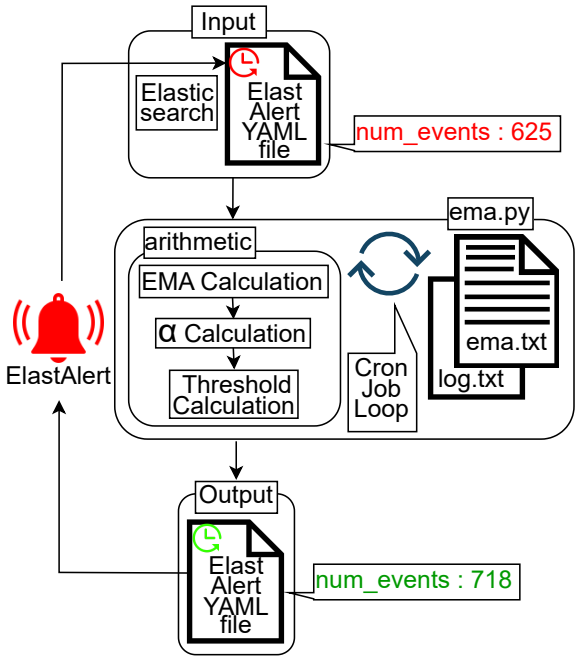

Figure 4: Code flow of ema.py

a dynamic threshold. ElastAlert receives the updated threshold from ema.py and applies the value to detect abnormal log activity. ElastAlert automatically sends alerts to the on-call monitor through Slack when warning syslog counts exceed the threshold. The process reduces manual monitoring effort for CDSL monitoring team members and improves the accuracy of alerting inside the CDSL environment.

4. Implementation

The implementation section integrates Elasticsearch, ElastAlert, and YAML rule files using one program named ema.py. The program updates threshold num_events inside ElastAlert YAML rule file automatically.

Table 1: Library configuration

Library	Version
PyYAML	6.0.2
elasticsearch-py	8.18.1

Table 1 shows the Library configuration. The implementation code uses the Python language as the main source code. The PyYAML library is included for enabling the processing of YAML configuration files inside ElastAlert. The elasticsearch-py library is for enabling the connection between elasticsearch to the source code for querying of warning logs.

Figure 4 shows the Code flow of ema.py. Input stage contains Elasticsearch and an ElastAlert YAML file with current num_events. The program reads warning log counts from Elasticsearch. The program also reads previous EMA and calculation history from ema.txt and log.txt. The arithmetic stage calculates EMA using Equation 1, calculates smoothing factor using Equation 2 and the dynamic threshold using Equation 3. A Kubernetes CronJob loop executes every hour and repeats the calculations. Output stage writes updated num_events, into ElastAlert YAML file. Inside Figure 4, the num_events metrics is updated from 625 to 718 based on the calculation of the EMA. Updated YAML file enables ElastAlert to apply a dynamic threshold for warning syslog counts inside CDSL environment.

Code 3 shows the Main function of ema.py. Line 1 defines the function named “main”. Line 2 prints a status message to indicate the start of log fetching. Line 3 calls “fetch_log_counts” and assigns hourly warning log counts to“counts”. Line 4 checks whether the number of elements in“counts” is smaller than n. Line 5 prints a message about insufficient logs for EMA calculation. Line 6 stops execution to avoid invalid calculations. Line 7 calls“load_last_ema” and loads the previous EMA value from ema.txt. Line 8 calculates a new EMA from“counts” using the previous EMA by calling“calculate_ema” based on Equation 1 and Equation 2. Line 9 computes the standard deviation of the most recent n counts for volatility measurement. Line 10 calculates the dynamic threshold using Equation 3.

Code 3: Main function of ema.py

```
1  def main():
2      print("Fetching logs from
       Elasticsearch...")
3      counts = fetch_log_counts()
4      if len(counts) < N:
5          print("Not enough logs to calculate
           EMA.")
6          return
7      prev_ema = load_last_ema()
8      ema = calculate_ema(counts,
       previous_ema=prev_ema)
9      stddev = statistics.stdev(counts[-N:])
10     threshold = ema + K * stddev
11     save_ema(ema)
12     update_yaml(threshold)
13     log_update(ema, stddev, threshold)
14     print(f "Updated threshold:
       {threshold:.2f}")
```

Line 11 saves the new EMA value into ema.txt with a timestamp. Line 12 writes the updated threshold into the ElastAlert YAML rule by setting num_events. Line 13 appends a log entry into log.txt containing EMA, standard deviation, and threshold for auditing. Line 14 prints the final threshold value as a confirmation message.

5. Evaluation

Experiment Environment

Table 2: Software configuration

Component	Version
Elasticsearch	8.13.4
ElastAlert2	2.11.1
Redmine	5.1.2

Table 2 shows the Software configuration. The versions of Elasticsearch, ElastAlert2, and Python are specified, together with the required Python libraries PyYAML and elasticsearch-py for YAML processing and Elasticsearch connection shown in Table 1. Redmine version 5.1.2 is used as the ticket management system to receive and manage alert notifications generated by ElastAlert.

Table 3 shows the Code files used in the experiment. The table includes the ElastAlert rule file, the Python script for EMA calculation, and log files that store the threshold history and EMA values.

Table 3: Code files used in the experiment

File	Description
warning.yaml	ElastAlert rule with dynamic threshold
ema.py	Python script to calculate and apply EMA threshold
log.txt	Historical logs of EMA, standard deviation, and thresholds
ema.txt	Historical EMA values for reference

Code 4: warning.yaml rule

```
1  name: High Warning syslog Rate
2  type: frequency
3  index: syslog-*
4  num_events: 322
5  timeframe:
6    hours: 1
```

Code 4 shows the “warning.yaml” rule. The rule defines the alert name, type, target index, and the threshold value under num_events. The configuration allows ElastAlert to detect when warning syslogs exceed the defined log count within one hour.

Code 5: log.txt entry

```
1  [2025-10-23T00:00:06.935862+00:00] EMA:
   624.96, StdDev: 68.51, Threshold: 727.72
```

Code 5 shows the log.txt entry. Each entry includes the timestamp, calculated EMA, standard deviation, and threshold value. The file provides a record of hourly threshold calculations for reference and verification.

Code 6: ema.txt entry

```
1  [2025-10-28T04:00:02.055632+00:00] EMA:
   261.48
```

Code 6 shows the ema.txt entry. The file stores the timestamp and the most recent EMA value calculated from the warning syslogs. The stored value is used as input for the next EMA calculation to maintain continuity between executions.

Table 4 shows the Virtual Machine resource configuration used to run the experiment. One virtual machine was used with 4 vCPU cores, 8 GB of memory, and 40 GB of storage capacity.

Table 4: Virtual Machine resource configuration

Resource	Specification
vCPU	4 cores
Memory	8 GB
Storage	40 GB

Basic Experiment

The basic experiment focuses on finding the most suitable variable values for the threshold calculation to obtain the most optimal results with the least amount of false and missed alerts. The variables are n and k values. The logs used in the experiment are the warning logs collected from September 2025 15 00:00 to September 22 2025 00:00. The syslog source is the Mint ESXi, one of the hypervisors inside the CDSL environment.

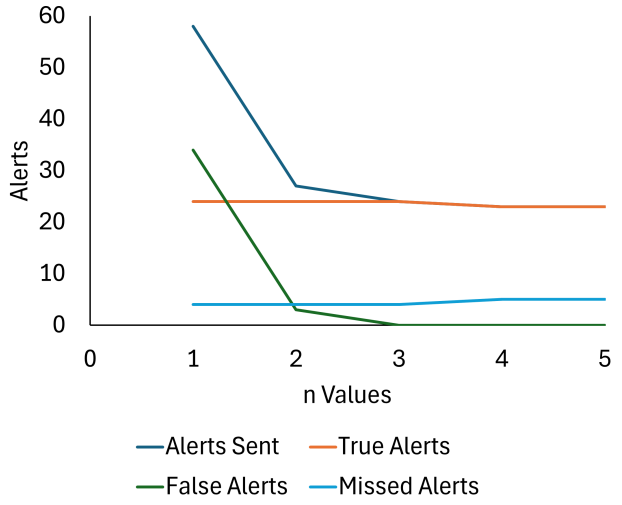

Figure 5: n value graph

Figure 5 shows the n value graph. The x-axis is the n Values which are 1 to 5, the y-axis is the number of Alerts. For the n value change, the k value is set to a constant of 1.5. The n value is changed from 1, 2, 3, 4, and 5. For n value 1, the number of alerts sent is 58, true alerts are 24, false alerts are 34, and missed alerts are 4. For n value 2, the number of alerts sent is 27, true alerts are 24, false alerts are 3, and missed alerts are 4. For n value 3, the number of alerts sent is 24, true alerts are 24, false alerts are 0, and missed alerts are 4. For n value 4, the number of alerts sent is 23, true alerts are 23, false alerts are 0, and missed alerts are 5. For n value 5, the number of alerts sent is 23, true alerts are 23, false alerts are 0, and missed alerts are 5. From the graph, the most optimal n value is 3 because it has 0 false alerts and the least number of missed alerts, 4, compared to $n = 4$ and 5 which has 5 missed alerts.

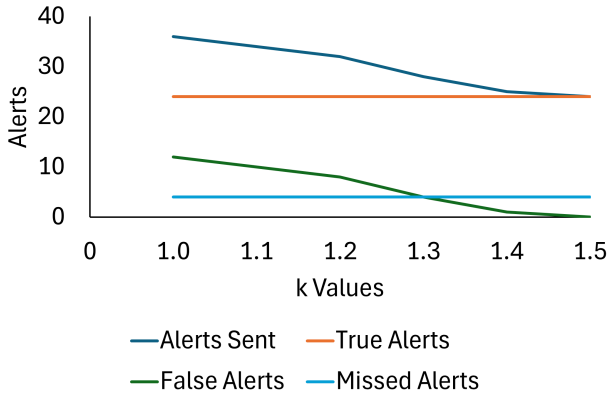

Figure 6: k value experiment

Figure 6 shows the k value experiment. For the k value experiment, the n value is set to a constant of 3. The k value changes from 1.0, 1.1, 1.2, 1.3, 1.4, to 1.5. For k value 1.0, the number of alerts sent is 36, true alerts are 24, false alerts are 12, and missed alerts are 4. For k value 1.1, the number of alerts sent is 34, true alerts are 24, false alerts are 10, and missed alerts are 4. For k value 1.2, the number of alerts sent is 32, true alerts are 24, false alerts are 8, and missed alerts are 4. For k value 1.3, the number of alerts sent is 28, true alerts are 24, false alerts are 4, and missed alerts are 4. For k value 1.4, the number of alerts sent is 25, true alerts are 24, false alerts are 1, and missed alerts are 4. For k value 1.5, the number of alerts sent is 24, true alerts are 24, false alerts are 0, and missed alerts are 4. From the graph, the most optimal k value is 1.5 because the number of false alerts is 0, compared to the smaller values which had false alerts.

Evaluation Experiment

The evaluation experiment integrates the current ElastAlert system used in the CDSL with the dynamic threshold mechanism from ema.py. The experiment applies the EMA-based calculation to the existing ElastAlert rule configuration. ElastAlert generates alerts and sends them to the CDSL on-call monitoring member for evaluation. The on-call monitor checks the alerts through monitoring applications to verify whether each alert represents an actual warning condition or a false alert. The experiment is conducted for 120 hours from October 29 2025, 10:00 AM to November 3 2025, 10:00 AM for the container logs and 96

hours from November 24 2025, 12:00 PM to November 27 2025, 12:00 PM for the hardware and hypervisor logs.

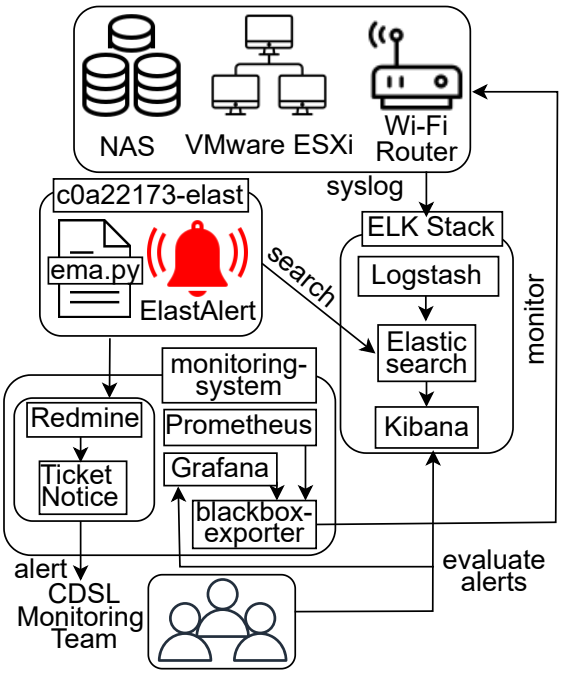

Figure 7: Flow of the evaluation experiment

Figure 7 shows the Flow of the evaluation experiment. The ELK Stack collects and visualizes syslogs through Logstash, Elasticsearch, and Kibana. ElastAlert which is set up in a VM called "c0a22173-elast" monitors the stored logs and sends alerts to Redmine using an API key for authentication. ElastAlert creates alert tickets and automatically notifies the CDSL on-call monitoring member. The on-call monitor receives the alert and checks system status through NAS-based monitoring applications, including Kibana, Grafana, and Prometheus. The verification process determines whether each alert is valid or false, providing information for calculating alert accuracy.

$$\text{Alert accuracy} = \frac{\text{True Alerts}}{\text{Total Alerts}} \tag{4}$$

Equation 4 shows the alert accuracy formula. The alert accuracy is calculated by dividing the number of true alerts by the total number of alerts sent during the evaluation period. The formula measures how accurately the EMA-based threshold detects real warning conditions while reducing false alerts.

$$\text{Average Alert Accuracy} = \frac{A_1 + A_2 + \cdots + A_m}{m} \tag{5}$$

Equation 5 calculates the average alert accuracy across all evaluated rules. Each A_i represents the alert accuracy percentage of one rule, and m is the total number of rules included in the evaluation. The formula provides a single overall accuracy value by averaging the percentages.

The following ElastAlert rules are used with the EMA-based dynamic threshold system for container logs, focusing on the "beats-*" index:

- **alertmanager-error rule**: Detects error-level logs generated by the Alertmanager component when alert delivery fails or the alert pipeline experiences internal errors.
- **prometheus-error rule**: Monitors error logs from Prometheus when target scraping fails or metric information cannot be fetched correctly.
- **prometheus-warning rule**: Detects warning-level logs from Prometheus related to temporary connection issues or metric timeout events.
- **dns rule**: Monitors syslog entries from the DNS server that indicate query resolution problems or abnormal request patterns.
- **dns-critical rule**: Detects critical-level logs from the DNS service, such as service crashes or complete resolution failures.
- **dhcp rule**: Monitors DHCP server logs that show IP allocation issues or lease conflicts in the network.
- **thanos-sidecar-flatline rule**: Detects inactivity of Thanos Sidecar by monitoring the absence of expected "upload new block" messages within a specific timeframe.

The following ElastAlert rules are used with the EMA-based dynamic threshold system for hardware and hypervisor logs, focusing on the "syslog-*" index:

- **rose-warning rule**: Monitors warning-level logs from the ESXi host named "rose," which runs multiple virtual machines used by students for experiments and research activities. The rule detects system warnings related to hardware performance or resource availability.
- **wifi-warning rule**: Monitors warning-level logs from the CDSL Wi-Fi router that provides internal and external network connectivity. The rule detects network connection instability, hardware signal degradation, or temporary packet loss events.
- **NAS-warning rule**: Monitors warning-level logs from the Network Attached Storage (NAS) used by ESXi hypervisors. The rule detects warnings related to storage read or write delays, file system issues, or temporary access failures.

Analysis of Evaluation Experiment Results

After the experiment period of 120 hours from October 29 2025, 10:00 AM to November 3 2025, 10:00 AM for the container logs and 96 hours from November 24 2025, 12:00 PM to November 27 2025, 12:00 PM for the hardware and hypervisor logs, ElastAlert sent several alerts for different rules. All alerts were registered to Redmine as tickets. Some rules did not generate any alerts during the experiment.

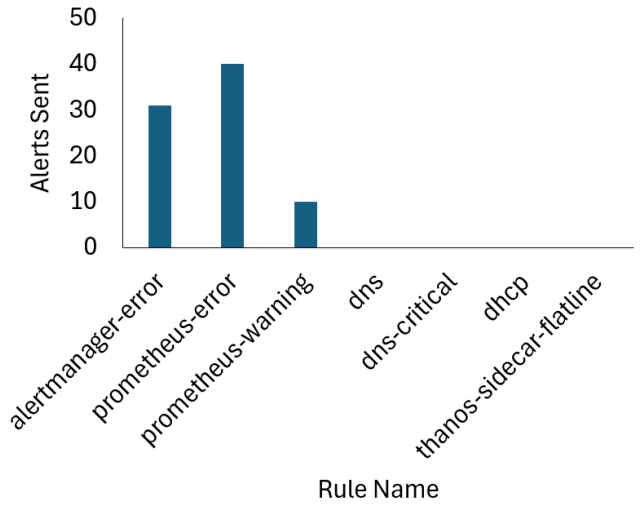

Figure 8: Container log alert count

Figure 8 shows the Container log alert count. The alertmanager-error rule sent 31 alerts, the prometheus-error rule sent 40 alerts, and the prometheus-warning rule sent 10 alerts. The dns, dns-critical, dhcp, and thanos-sidecar-flatline rules did not send any alerts. The difference in alert counts reflects the operational activity and log frequency of each monitored service. For the evaluation of alerts sent, only a limited number of alerts were reviewed by the CDSL monitoring team due to variations in monitoring shift times, weekends, and the total number of alerts generated.

Table 5 shows the Container log partial alert accuracy. The table includes the warning log count for each rule, which represents the number of warning severity logs observed during the evaluation period. A higher warning log count indicates more frequent warning events, which increases the chance of generating alerts. A total of 41 alerts were evaluated, 13 from the alertmanager-error rule, 19 from the prometheus-error rule, 6 from the prometheus-warning rule, and 3 from the thanos-sidecar-flatline rule. Based on Equation 4, the alert accuracy values are 6/13 (46.2%) for the alertmanager-error rule, 7/19 (36.8%) for the prometheus-error rule, and 4/6 (66.7%) for the prometheus-warning rule. The results show that the prometheus-warning rule had the highest accuracy, while the other rules showed moderate or low accuracy.

Table 5: Container log partial alert accuracy

Alert Rule	alertmanager-error	prometheus-error	prometheus-warning
Warning Log Count	1687	66	43
Alert Count	31	40	10
Evaluated Alerts	13	19	6
True Alerts	6	7	4
False Alerts	7	12	2
Alert Accuracy	46.2%	36.8%	66.7%

alertmanager-error rule

Code 7: alertmanager-error true alert log

```
1 ts=2025-11-26T07:20:41.904Z
  caller=dispatch.go:353 level=error
  component=dispatcher msg="Notify for alerts
  failed" num_alerts=1
  err="redmine/webhook[1]: notify retry
  canceled due to unrecoverable error after 1
  attempts: unexpected status code 404:
  http://c0117304-test:5005/webhook:
  {\"detail\":\"Not Found\"}"
```

Code 7 shows the alertmanager-error true alert log. The log contains the message “Notify for alerts failed” and reports a 404 error, which indicates that Alertmanager could not deliver the alert. The failure represents a real notification problem in the alert pipeline. The alert was double checked using Prometheus and Grafana to confirm that the system experienced an actual delivery failure.

Code 8 shows the alertmanager-error false alert log. The message indicates that the notification was sent successfully and mentions only a minor formatting issue. The alert does not represent a failure in Alertmanager functionality.

Code 8: alertmanager-error false alert log

```
ts=2025-11-27T06:58:13.441Z
caller=notify.go:274 level=info
component=dispatcher msg="Notification sent"
alertname=InternalServiceDown
detail="Delivery completed with minor
formatting error: missing field
'service_group'."
```

prometheus-error rule

Code 9: prometheus-error true alert log

```
ts=2025-11-27T23:59:59.951Z
caller=scrape.go:1288 level=error
component="scrape manager"
scrape_pool=Internal-archive-node-exporter
target=http://192.168.100.31:9100/metrics
msg="Scrape commit failed" err="write to
WAL: log samples: write
/prometheus/wal/00001196: no space left on
device"
```

Code 9 shows the prometheus-error true alert log. The log reports "Scrape commit failed" and indicates that Prometheus could not write data because the device had no space left. The message reflects a real scraping failure that affects metric collection. The alert was double checked using Prometheus and Grafana and was verified as a valid system issue.

Code 10: prometheus-error false alert log

```
ts=2025-11-26T08:59:56.782Z
caller=group.go:518 level=warn
name="Internal Monitoring Cluster
PersistentVolumeFullSoon" index=11
component="rule manager"
file=/etc/prometheus/rules/monitoring-
cluster.yml
group=Internal-Monitoring-Cluster-Check
msg="Skipped rule evaluation due to
temporary metadata parsing issue"
```

Code 10 shows the prometheus-error false alert log. The message reports a rule evaluation warning but does not indicate a scraping failure or service problem. Prometheus continued operating normally, and the warning was unrelated to system health.

Code 11: prometheus-warning true alert log

```
ts=2025-10-29T04:06:01.299Z
caller=group.go:492 level=warn
name="Internal Core Server Cluster Node CPU
Usage High 80" index=9 component="rule
manager"
file=/etc/prometheus/rules/core-server-
monitoring-rules.yml
group=Internal-Core-Server-Check
msg="Evaluating rule failed" rule="alert:
Internal Core Server Cluster Node CPU Usage
High 80\nexpr: avg without (cpu)
(rate(node_cpu_seconds_total{instance=~\
"192.168.100.35:9100|192.168.100.36:9100|
192.168.100.37:9100|192.168.100.38:9100\",
mode!=\"idle\"}[5m])) * 100 >= 80\nfor:
3m\nlabels:\n severity:
warning\nannotations:\n alert_title: 'Core
Server Cluster Node CPU usage > 80%: {{
$labels.instance }}'\n description: 'CPU
usage exceeds 80%: {{ $labels.instance }}'\n
runbook_url:
https://cdsl-tut.esa.io/posts/3219\n
summary: Node {{ $labels.instance }} CPU
usage is over 80%\n" err="vector cannot
contain metrics with the same labelset"
```

prometheus-warning rule

Code 11 shows the prometheus-warning true alert log. The log reports "Evaluating rule failed" for a CPU usage alert, which indicates that Prometheus could not process the rule correctly. The failure affected the monitoring of CPU usage across nodes. The alert was double checked using Redmine to confirm that the system experienced an actual delivery failure.

Code 12: prometheus-warning false alert log

```
ts=2025-11-25T14:12:33.552Z
caller=manager.go:211 level=warn
component="rule manager" msg="Ignoring
malformed label" detail="invalid label
format: unexpected character '=' in label
name"
```

Code 12 shows the prometheus-warning false alert log. The message indicates a warning caused by a malformed label inside a rule file, which does not affect metric scraping or system behavior. Prometheus continued to operate normally, and the warning was only related to parsing an incorrect label format. The alert was double checked using Prometheus and Grafana and was confirmed to be unrelated to any real monitoring

problem.

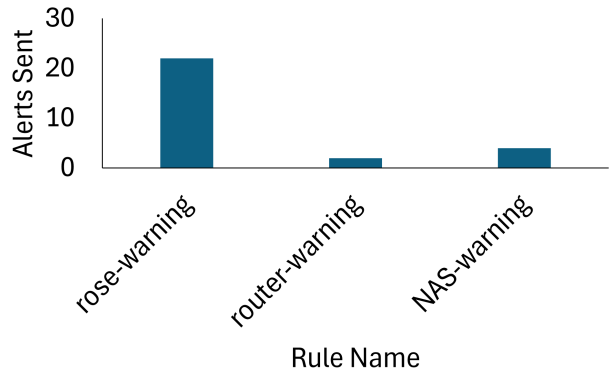

Figure 9: Hypervisor and hardware log alert count

Figure 9 shows the Hypervisor and hardware log alert count. The rose-esxi-warning rule sent 22 alerts, the router-warning rule sent 2 alerts, and the NAS-warning rule sent 4 alerts. The counts reflect the operational state of each hardware component during the monitored period. For the evaluation of alerts sent, only a partial number of alerts were reviewed by the CDSL monitoring team due to variations in monitoring shift times, weekends, and the total number of alerts generated.

Table 6: Hypervisor and hardware log partial alert accuracy

Alert Rule	**rose-warning**	**wifi-warning**	**NAS-warning**
Warning Log Count	111	2	11
Alert Count	22	2	4
Evaluated Alerts	14	2	4
True Alerts	0	0	0
False Alerts	14	2	4
Alert Accuracy	0.0%	0.0%	0.0%

Table 6 shows the Hypervisor and hardware log partial alert accuracy. The table includes the warning log count for each rule, which represents the number of warning severity logs observed during the evaluation period. A higher warning log count indicates more frequent warning events, and a lower count reflects fewer opportunities for alerts to be triggered. A total of 20 alerts were evaluated: 14 from the rose-warning rule, 2 from the wifi-warning rule, and 4 from the NAS-warning rule. All evaluated alerts were classified as false, so the alert accuracy for every hypervisor and hardware rule is 0.0%. The results indicate that the EMA-based threshold and filter conditions did not work effectively for syslog-level warnings from these devices.

rose-warning rule

Code 13: rose-warning false alert log

```
<12>2025-11-01T06:13:38.334Z
rose.a910.tak-cslab.org
ConfigStore[2099016]: warn
[ConfigStore:a435b9a3c0] No tasks generated
for the desired configuration;
profile="HostConfigProfile",
version="7.0.3-20842708", module="hostd",
opID="995e7c9b", detail="requested state
identical to current state"
```

Program 13 shows the rose-warning false alert log. The message is a configuration store warning that reports that no tasks were generated because the requested state was identical to the current state. The log does not describe any hardware fault or performance degradation on the ESXi host. The alert was checked by accessing the “rose” ESXi hypervisor and by reviewing related metrics and alerts on Prometheus, and no abnormal CPU, memory, or storage behavior was observed. The checks confirmed that this log represents a harmless configuration message, so the alert was classified as false.

wifi-warning rule

Code 14: wifi-warning false alert log

```
<12>Nov 2 22:44:51 TUF-AX5400-DFD8-AAB1AD3-C
kernel: [wl0.1] sta_info.c:3214: MAC
00:0C:29:F3:16:7F not found in UDB, skipping
removal; reason=STA_ENTRY_MISSING, rssi=-57,
vap=wl0.1, ifname=br0
```

Program 14 shows the wifi-warning false alert log. The kernel message reports that a specific MAC address

is missing from the internal table and is skipped during removal. The message describes an internal lookup event and does not indicate packet loss, throughput degradation, or client disconnection. The alert was checked by confirming the Wi-Fi router status and by reviewing network-related metrics and alerts on Kibana and Prometheus. The checks showed that users did not experience connectivity problems, so the alert was judged to be a false warning.

NAS-warning rule

Code 15: NAS-warning false alert log

```
<14>1 2025-10-30T17:35:32+09:00 tapioca
System - - [synolog@6574 synotype="System"
luser="cdsl" event="System successfully
created
[k8s-csi-pvc-2e292144-a851-440f-98c9-
649444647088](Device Type is [BLUN]) on
[/volume1]."][meta sequenceId="7"] cdsl:
System successfully created
[k8s-csi-pvc-2e292144-a851-440f-98c9-
649444647088](Device Type is [BLUN]) on
[/volume1].
```

Program 15 shows the NAS-warning false alert log. The message reports that a storage object for a Kubernetes persistent volume was created successfully on /volume1 and clearly states that the operation completed normally. The log does not contain any error code or failure description. The alert was checked using the NAS manager to confirm that the volume and LUN status were normal and by reviewing storage-related metrics and alerts on “Prometheus”. The checks showed that the event represented a normal creation process, so the alert was treated as a false warning.

Alert Accuracy

The alert accuracy for each rule is calculated using Equation 4. The alertmanager-error rule recorded an accuracy of approximately 46.2% (6/13), the prometheus-error rule recorded approximately 36.8% (7/19), and the prometheus-warning rule recorded approximately 66.7% (4/6). The rose-warning rule recorded an accuracy of 0.0% (0/14), the wifi-warning rule recorded 0.0% (0/2), and the NAS-warning rule recorded 0.0% (0/4) because all evaluated alerts for these rules were classified as false. The overall alert accuracy is calculated using Equation 5. The average alert accuracy is calculated as;

$$\frac{0.462 + 0.368 + 0.667 + 0.000 + 0.000 + 0.000}{6} \approx 0.2495$$

The value indicates that the EMA-based threshold achieved reasonable accuracy for container logs but did not perform well for hypervisor and hardware logs.

The results occurred because the filters inside the ElastAlert YAML files were not specific enough to match the actual warning conditions shown in the log examples. Several logs were labeled as false alerts because the filters captured messages that contained warning-level keywords but did not represent real system problems. The false alert examples show that many messages included harmless warnings, normal operational events, or minor formatting issues that should have been excluded by the filter conditions. The results indicate that the alert accuracy depends strongly on the quality of the filtering rules. More precise filters can reduce the number of unrelated logs, prevent unnecessary alerts, and improve the reliability of the EMA-based threshold in the CDSL environment.

6. Discussion

The current ElastAlert configuration in the CDSL environment uses manually defined filters created by monitoring members based on past alert logs. Manual filter definition limits adaptability and reduces alert accuracy because the selected filters sometimes mismatch. A clustering-based method using K-Means and DBSCAN should be used to solve this problem [13]. The method groups logs automatically and finds unusual patterns without manual setup. Using this method to set filters automatically improves alert accuracy and reduce the need for human-defined filters.

The current implementation of ElastAlert filters logs directly from Elasticsearch without using a dedicated database for log storage. This approach limits the ability to perform detailed analysis and makes it difficult to apply advanced filtering based on historical data. A database-based log filtering system should be used to solve this problem [14]. The method stores warning logs in a database, allowing the system to perform structured queries and analyze past patterns more efficiently. A graph-based or collaborative filtering approach can also be applied to the stored logs to support prediction and classification during future filtering. Storing logs in a database improves log management and makes future filtering and analysis faster and more accurate.

The current EMA-based threshold calculation depends only on past log counts. When a similar anomaly happens again, the EMA value does not compare new anomalies with past log counts, and alert accuracy decreases. A Bayesian threshold estimation method should be used to solve this problem [15]. The method updates the threshold using past logs and new observations, making the threshold follow changes in log behavior more accurately. Using Bayesian analysis in ElastAlert improves alert accuracy when the system logs show repeating patterns.

7. Conclusion

This study proposes a method to determine a dynamic threshold for log monitoring using the Exponential Moving Average (EMA) to improve alert accuracy in "ElastAlert". The system collects warning syslogs from "Elasticsearch" and calculates the EMA, the standard deviation, and a dynamic threshold that replaces the fixed num_events value inside the ElastAlert YAML configuration file. The method adapts the threshold to changes in syslog activity and reduces false alerts caused by static thresholds. The basic experiment tested different values of n and k, and the results showed that $n = 3$ and $k = 1.5$ provided the most stable performance with the least number of false and missed alerts. The evaluation experiment applied the dynamic threshold to the current CDSL ElastAlert system for both container logs and hypervisor or hardware logs. The alert accuracy for container logs was approximately 46.2% for the alertmanager-error rule, approximately 36.8% for the prometheus-error rule, and approximately 66.7% for the prometheus-warning rule. The alert accuracy for hypervisor and hardware logs was 0.0% for the rose-warning rule, 0.0% for the wifi-warning rule, and 0.0% for the NAS-warning rule. The average alert accuracy across all rules was approximately 24.9%. The results show that the EMA-based threshold improved alert behavior for container logs but did not perform well for hypervisor or hardware logs. The failure occurred because the filter conditions inside the ElastAlert YAML files did not match the actual warning patterns in the syslogs. The filters captured many unrelated messages, which increased false alerts and reduced accuracy. The method followed log trends correctly and adjusted threshold sensitivity based on real-time activity, but precise filtering was necessary to apply the threshold effectively. Overall, the EMA-based dynamic threshold increased adaptability and improved alert reliability compared to static thresholds for container-based monitoring inside the CDSL environment. Further refinement is required for syslog-level warnings, especially for logs from ESXi hosts, NAS devices, and Wi-Fi routers, where more accurate filter definitions are needed to prevent false alerts and capture meaningful warning conditions.

References

[1] Gerhards, R.: The Syslog Protocol, RFC 5424 (2009).
[2] Lonvick, C. M.: The BSD Syslog Protocol, RFC 3164 (2001).
[3] Rukhsar, K. S.: Real-Time Data Monitoring System Using Elk Stack (Elasticsearch, Logstash, Kibana), *International Journal of Multidisciplinary Research in Science, Engineering, Technology and Management*, Vol. 10, No. 4 (2023).
[4] Rochim, A. F., Aziz, M. A. and Fauzi, A.: Design Log Management System of Computer Network Devices Infrastructures Based on ELK Stack, *2019 International Conference on Electrical Engineering and Computer Science (ICECOS)*, pp. 338–342 (online), DOI: 10.1109/ICECOS47637.2019.8984494 (2019).
[5] Bakar, N., Belaton, B. and Samsudin, A.: False positives reduction via intrusion alert quality framework, *2005 13th IEEE International Conference on Networks Jointly held with the 2005 IEEE 7th Malaysia International Conf on Communic*, pp. 6 pp.– (online), DOI: 10.1109/ICON.2005.1635545 (2005).
[6] Liu, H.-b., Hao, Y.-h. and Zuo, Z.: Design and Implementation of Data Acquisition System Based on Dynamic Threshold, *2019 International Conference on Communications, Information System and Computer Engineering (CISCE)*, pp. 555–558 (online), DOI: 10.1109/CISCE.2019.00129 (2019).
[7] Huang, X., Wang, Z., Ai, Y. and Feng, J.: Application and Research Based on Exponential Weighted Moving Averages and GRU Modeling, *2024 IEEE 2nd International Conference on Sensors, Electronics and Computer Engineering (ICSECE)*, pp. 795–800 (online), DOI: 10.1109/ICSECE61636.2024.10729530 (2024).
[8] Stoleriu, R., Puncioiu, A. and Bica, I.: Cyber Attacks Detection Using Open Source ELK Stack, *2021 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pp. 1–6 (online), DOI: 10.1109/ECAI52376.2021.9515120 (2021).
[9] Ghafouri, A., Abbas, W., Laszka, A., Vorobeychik, Y. and Koutsoukos, X.: Optimal thresholds for anomaly-based intrusion detection in dynamical environments, *ACM Transactions on Cyber-Physical Systems* (2016).
[10] Bobde, H.: Log Alert System: Server Log Recognition and Alert System, *International Journal of Trend in Scientific Research and Development (IJTSRD)*, Vol. 7, No. 5, pp. 1045–1048 (online), DOI: 10.31142/ijtsrd70555 (2023).
[11] Yang, H., Zhang, G., Su, Y. and Guo, N.: A power fusion data cleaning method based on exponential moving average and cosine similarity algorithms, *2024 IEEE 10th International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pp. 25–30 (online), DOI: 10.1109/EdgeCom62867.2024.00012 (2024).
[12] Prami Swari, M. H., Susila Handika, I. P. and

Susila Satwika, I. K.: Comparison of Simple Moving Average, Single and Modified Single Exponential Smoothing, *2021 IEEE 7th Information Technology International Seminar (ITIS)*, pp. 1–5 (online), DOI: 10.1109/ITIS53497.2021.9791516 (2021).

[13] Mishra, A. K., Bagla, P., Sharma, R., Pandey, N. K. and Tripathi, N.: Anomaly Detection from Web Log Data Using Machine Learning Model, *2023 7th International Conference on Computer Applications in Electrical Engineering-Recent Advances (CERA)*, pp. 1–6 (online), DOI: 10.1109/CERA59325.2023.10455153 (2023).

[14] Kuwano, M., Okuma, M., Okada, S. and Mitsunaga, T.: ATTCK Behavior Forecasting based on Collaborative Filtering and Graph Databases, *2022 IEEE International Conference on Computing (ICOCO)*, pp. 191–197 (online), DOI: 10.1109/ICOCO56118.2022.10032036 (2022).

[15] Nakajima, J. and West, M.: Bayesian Analysis of Latent Threshold Dynamic Models, *Journal of Business & Economic Statistics*, Vol. 31, No. 2, pp. 151–164 (online), DOI: 10.1080/07350015.2012.747847 (2013).

