Adaptive Threshold Determination for Syslog Monitoring by Exponential Moving Average

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

Abstract：The syslog is used to convey event notification messages. The event notification messages include warning-level logs which show when there is a system-related problem. Conventional alerting systems like Elastalert rely on fixed thresholds. The fixed thresholds are vulnerable to sudden changes in the log count. Factors such as VM usage, time of day, and hardware faults cause fluctuations in log volume, making static thresholds prone to false or missed alerts. The on-call engineer monitors Kibana to track system logs. This paper proposes a method for dynamically updating alert thresholds for syslog's warning-level logs using Exponential Moving Average (EMA). A script was developed in Python to fetch warning-level log counts from Elasticsearch, compute the threshold hourly, and update the Elastalert YAML rule file accordingly. The EMA-based method was compared with human-defined fixed thresholds over a 72-hour period and showed improved adaptability with less alerts compared to the fixed thresholds in the experiment. The live evaluation experiment was done from 12:00 PM, July 7, 2025 to 12:00 PM, July 9, 2025 (JST). The alert number was 11, of which 9 were true and 2 were false, achieving approximately 81.8 % accuracy. The results demonstrate that the EMA-based method reduces false alerts while maintaining high adaptability to the change in log counts, making it a reliable alternative for real-time log monitoring.

1. Introduction

Background

The syslog is a record that contains information about events, errors, or operations inside computer systems and network devices [1]. Syslogs include data such as timestamps, log levels, messages, and the name of the process that created the log. System administrators use logs to check system status, investigate problems, and analyze past activities [2].

Figure 1 shows the Syslog and ELK stack architecture. The ELK Stack consists of Elasticsearch, Logstash, and Kibana for log collection, storage, and visualization. Logstash receives the syslogs from NAS, Machines in VMware ESXi, and Wi-Fi Router and forwards the logs to Elasticsearch. Elasticsearch stores and indexes the syslogs. Kibana connects to Elasticsearch and shows the syslog contents on dashboards. The ELK stack structure makes log monitoring easier and faster [3, 4].

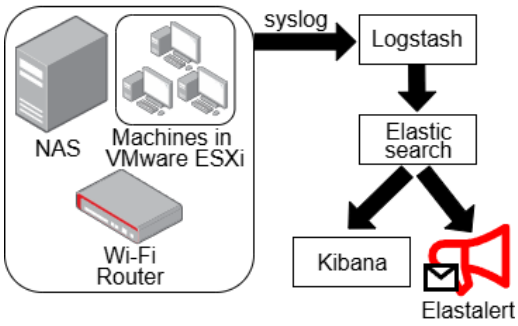

Figure 1: Syslog and ELK stack architecture

Log monitoring allows the system operator to detect problems early, before they lead to system failures or service disruptions. The monitoring process also helps prevent system failure and reduce system downtime. The system operator takes quick action when the monitoring dashboard shows warning signs or abnormal activity.

Figure 2 shows the monitoring process using Elastalert.

[1] Tokyo University of Technology, Department of Computer Science
1404-1 Katakuracho, Hachioji City, 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

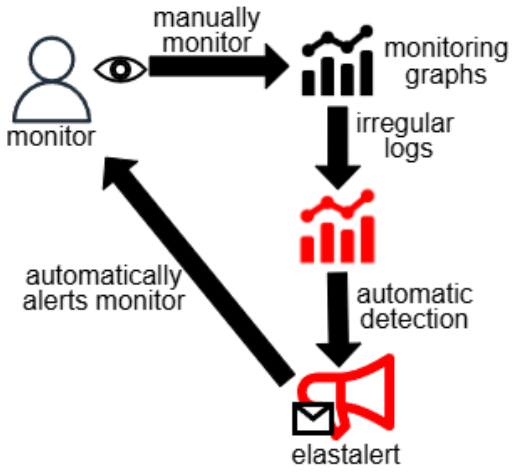

Figure 2: Monitoring process using Elastalert

The monitor watches the monitoring graphs. When the number of warning logs increases, Elastalert automatically detects the change. Elastalert sends an alert to the monitor. The alert system helps the on-call monitor respond to problems quickly and reduces the chance of missing events [5].

Main Issue

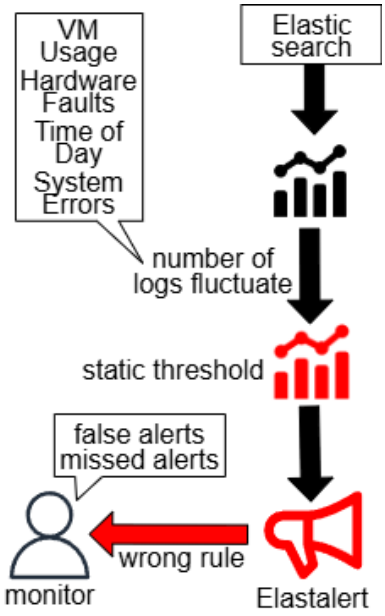

Figure 3: Static threshold leads to false and missed alerts.

Figure 3 shows the static threshold leads to false and missed alerts. The number of warning logs stored in Elasticsearch fluctuates based on different factors. The number changes depending on various factors such as VM usage load, time of day, network activity, hardware faults, and system errors. These fluctuations create dynamic log patterns that make it difficult to define a fixed threshold for alerting.

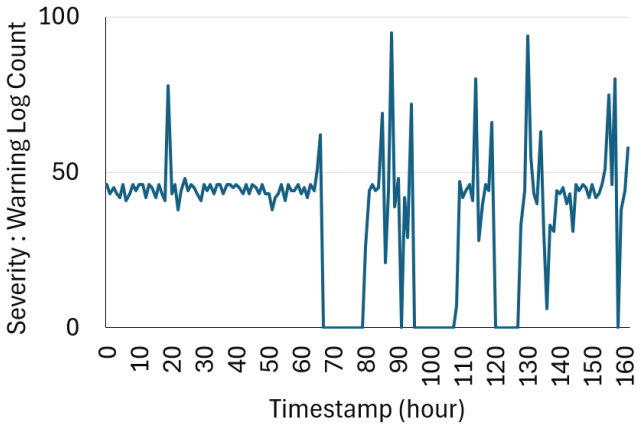

Figure 4: Fluctuation of warning severity logs

Figure 4 shows the fluctuation of warning severity logs. The y-axis is the warning severity log cunt, and the x-axis shows the timestamp of the logs, recorded hourly. Throughout the duration of 168 hours, from 12:00 AM, 8 June, 2025 to 12:00 AM, 15 June, 2025 (JST), the log count has increased and decreased drastically numerous times, seen in Figure 4. The Figure shows that depending on various factors, a static threshold is prone to sending false alerts.

Code 1: YAML File Metrics

```
type: frequency
num_events: 410
timeframe:
  minutes: 60
```

Code 1 shows the YAML file metrics. This code is a part of the Elastalert YAML configuration file. The required rule configurations which are `num_events` and `timeframe` set the number of log counts (`num_events`) in a certain timeframe (`timeframe`) to send an alert. Because the volume and pattern of logs change frequently, especially across different times and devices, setting a static value for these metrics becomes unreliable based on Figure 4. A threshold that works well during normal hours is strict during peak usage and lenient when something abnormal actually occurs. The mismatch between static thresholds and fluctuating log number causes two major problems:

- **False Alerts:** Non-critical or expected log spikes exceed the threshold and trigger unnecessary alerts, overwhelming the on-call monitor [6].
- **Missed Alerts:** When the log activity genuinely indicates an anomaly, the static rule does not recognize

it because the threshold is high or outdated, resulting in no alert being sent [6].

The system's current inability to adapt to real-time log trends reduces the trustworthiness of the alerts and increases the workload on the monitoring team. The issue becomes more severe when dealing with logs from various sources like VMware ESXi, NAS, and Wi-Fi routers, each of which has unique patterns. The system needs to achieve both sensitivity and accuracy in alerting. A threshold that meets the two conditions should be determined.

Overview of Each Section

Section 2 describes related research. Section 3 describes the proposed method and use case scenarios. Section 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 Research

Previous work has explored alert systems in various domains. One study proposed the use of email-based alerts to detect errors in hospital logs [2]. While the approach was effective for medical systems, it used basic rule types and did not address complex or dynamic log environments. In contrast, this study targets infrastructure-level monitoring using multiple Elastalert rule types and evaluates them based on the pattern of logs from sources such as VMware ESXi, NAS, and Wi-Fi routers.

Other research has implemented the ELK Stack, Elasticsearch, Logstash, and Kibana to support real-time log data monitoring and visualization [3]. These systems help users understand log patterns through dashboards. The system lacks automation in alertingThis study extends the log monitoring by integrating Elastalert to automate notifications and reduce manual monitoring. This study also focuses on a specific index, the syslog compared to using the whole ELK stack [1].

Additional work has applied Exponential Moving Average (EMA) in the context of time-series data processing, particularly for data cleaning and smoothing [7]. Although the use of EMA in that study focused on improving prediction accuracy in power data, the concept of dynamic adjustment is shared. This study applies EMA to real-time log monitoring by dynamically adjusting alert thresholds in Elastalert, aiming to reduce false alerts and improve anomaly detection accuracy.

3. Proposal

This study aims to make Elastalert dynamically change its alert threshold according to changes in syslog counts. CDSL members sees false alerts on the Slack channel which is integrated with the Elastalert webhook. The false alerts are caused by the log.syslog.severity.name : "Warning" log count fluctuating based on various factors shown in Figure 3. These false alerts confuse the on-call monitor and reduce trust in the monitoring system [8].

3.1 Using EMA (Exponential Moving Average) to Determine Rule Metrics

Exponential Moving Average (EMA) is a method that calculates the average of values over time. EMA gives more importance to the latest data and less to the older data unlike a normal average [9]. The EMA helps the system detect changes while still keeping track of past patterns. EMA is used in this paper because it adapts to the changes in log counts, unlike a fixed threshold.

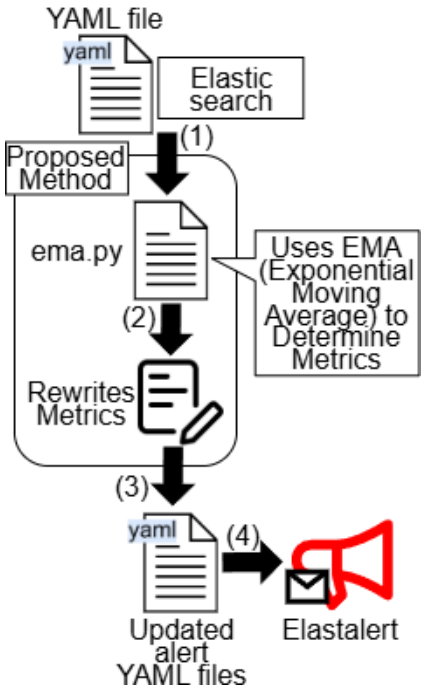

Figure 5: Overview of the proposed method

Figure 5 shows the overview of the proposed method. The system first reads the log data from Elasticsearch. ema.py is the Python script. The Python script then calculates the EMA value using past log counts per every hour. The script updates the alert threshold value in the YAML file after calculating the EMA. The updated YAML file is passed to Elastalert, which uses the new

threshold to decide when to send alerts [10].

The EMA formula makes the threshold change adaptively. When the log count goes up suddenly, the EMA also goes up, making the threshold increase. When the log count becomes low, the EMA slowly goes down, making the threshold decrease. The method allows the alert threshold to follow the real pattern of logs. The threshold becomes dynamic instead of fixed. The threshold helps avoid alerts when the log level is normal. Elastalert sends an alert when an observed value exceeds the threshold. The threshold improves the alerting accuracy and reduces stress for the on-call monitor [7, 11].

3.1.1 EMA Formula

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

- EMA_t : The current EMA (Exponential Moving Average) value at time t.
- x_t : The current log count value at time t.
- α : The smoothing factor that controls how much weight is given to the most recent data.
- EMA_{t-1} : The previous EMA value from time $t-1$.

Equation (1) shows the formula for calculating the Exponential Moving Average. Equation (1) uses the current log count value x_t and the previous EMA value EMA_{t-1} to calculate the new EMA value EMA_t. The smoothing factor α controls how much the new value depends on the recent data. A larger α makes the EMA follow the latest log count more closely, while a smaller α makes the EMA respond more slowly [12, 13].

$$\alpha = \frac{2}{n+1} \qquad (2)$$

- n : The time smoothing period.

Equation (2) shows the formula for calculating the smoothing factor α. Equation (2) for α adjusts how sensitive the EMA is to recent data. A smaller value of n produces a larger α, which makes the EMA follow recent log changes more quickly. A larger value of n results in a smaller α, which makes the EMA change more slowly [12].

$$\text{Threshold} = \mathrm{EMA}_t + k \cdot \sigma \qquad (3)$$

- k : A constant value that adjusts the strictness of the alert threshold.
- σ : The standard deviation of recent log values. The number of log values used is determined by n.

Equation (3) shows the formula for calculating the dynamic alert threshold. Equation (3) adds a margin to the EMA value by including $k \cdot \sigma$. When the standard deviation is large, the threshold becomes higher, making the alert rule more tolerant. When the log values are stable, the threshold stays close to the EMA. The EMA method helps reduce false alerts and adapts the alert rule to the actual pattern of the system [11].

3.1.2 k and n Values

Table 1 shows the Smoothing factors α for different n values. When changing the value of n, the pattern of the EMA formula also changes. The value of n controls the time smoothing period. Log data with one hour intervals are used to decide the n value. For example, when $n = 3$, the EMA calculation uses the trend of the past 3 hours. A smaller n results in a larger α, which makes the EMA respond quickly to sudden changes. A larger n gives a smaller α, which makes the EMA respond more slowly and smooth out short spikes.

Table 1: Smoothing factors α for different n values

n (Smoothing Period)	α (Smoothing Factor)
3	0.500
5	0.333
10	0.182
15	0.125
20	0.095

Figure 6 shows the n value graph. The log count is taken from 12:00 AM, 8 June, 2025 to 12:00 AM 10 June 2025 (JST). The value of n affects how fast the EMA responds to changes in log counts. Each EMA line uses a different n value, as listed in Table 2. The k value is set at a constant of $k = 1.5$. When n is small, the EMA reacts quickly to a sudden spike in log counts. The values make the threshold more responsive, and it also leads to more frequent alerts. When n is large, the EMA changes more slowly. The values make the threshold more stable and reduce false alerts, and delay the detection of sudden changes. Therefore, smaller n values are good for fast detection, while larger n values are better for avoiding noise.

Table 2 shows the Number of alerts sent for each n Value in one week, based on different values of n. As the n value increases, the number of alerts sent decreases. A higher n value results in a more responsive alert threshold. A faster response causes more log counts to exceed the threshold, so more alerts are triggered. A lower n value is more sensitive and allows alerts to be sent more easily.

Figure 7 shows the k value graph. The log count is taken from 12:00 AM, 8 June, 2025 to 12:00 AM 10 June 2025

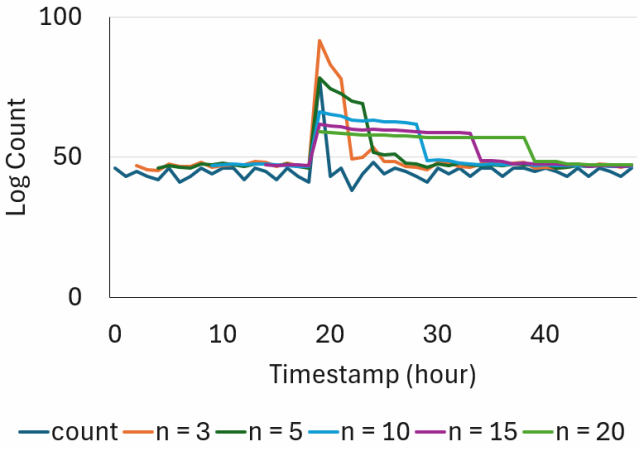

Figure 6: N Value Graph

Table 2: Number of alerts sent for each n Value in one week

N Value	Alerts Sent
3	23
5	20
10	16
15	16
20	9

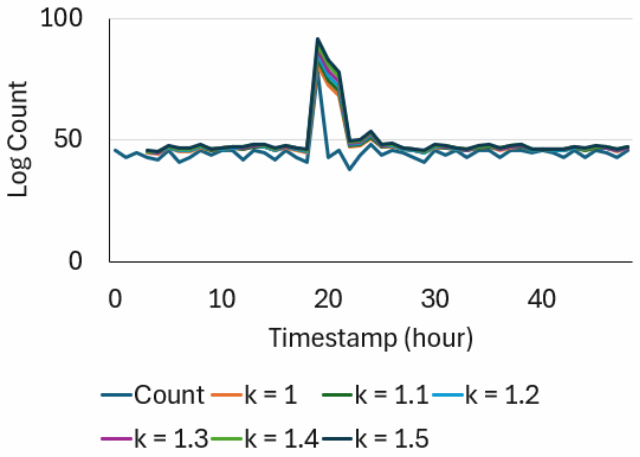

Figure 7: K Value Graph

(JST). The value of k affects the alert threshold based on the same data range used for the n value. The n value is set to a constant of 3. k is a constant value that controls how high the threshold is above the EMA based on Equation (3). When k is small, the threshold is close to the EMA and follows it closely. As k increases, the threshold moves further away from the log count and becomes harder to exceed.

Table 3 shows the number of alerts sent for each k value for one week. As the k value increases, the number of alerts sent decreases. A higher k results in a higher alert threshold. Less number of log counts exceed the threshold, so less alerts are triggered with a higher threshold. The value of k is set to 1.5 and n is set to 3 based on the results of both n and k values.

Table 3: Number of alerts sent for each k value for one week

k Value	Alerts Sent
1.0	40
1.1	38
1.2	36
1.3	30
1.4	27
1.5	25

Use Case Scenario

The CDSL (Cloud and Distributed Systems Laboratory) Monitoring Team checks the system logs to make sure all devices are working correctly. The team uses tools like NAS, Grafana, and Kibana. Elastalert is used to send alerts automatically when an anomaly occurs to reduce the work of watching dashboards all the time.

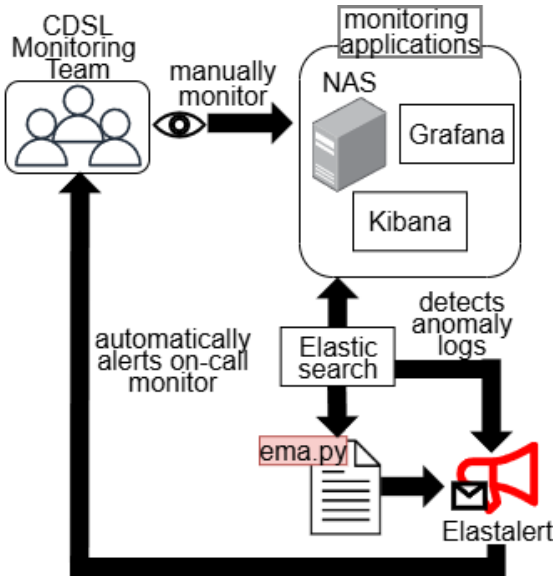

Figure 8: Use Case Scenario

Figure 8 shows the Use Case Scenario. The CDSL monitoring team monitors system logs using several applications, such as NAS, Grafana, and Kibana. Kibana gets its log data from Elasticsearch. Logstash receives data from devices in the system and sends it to Elasticsearch. Elastalert checks the data inside Elasticsearch. When

Elastalert finds something abnormal, it sends an alert to the on-call monitor. The process makes the monitoring easier because the team does not need to keep watching the dashboard all the time.

4. Implementation

Code Implementation

The Python file `ema.py` is used to fetch log data from Elasticsearch, calculate the EMA, and set a dynamic threshold for the `num_events` value inside the Elastalert YAML rule file. The code uses the Python packages `elasticsearch`, `yaml`, `datetime`, and `statistics`.

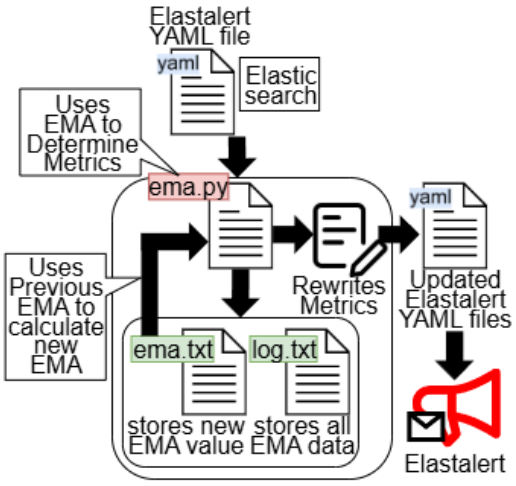

Figure 9: Code Flow of ema.py

Figure 9 shows the code flow of `ema.py`. The script first retrieves log data from Elasticsearch for a specific hostname and severity level, then calculates the total number of logs per hour for the past 7 days. The script computes the EMA using the proposed formula and adds a margin using the standard deviation and constant k to generate a dynamic threshold. The threshold is inserted into the YAML rule file under the `num_events` key, as shown in Code 1. The script also saves the EMA and threshold history to `ema.txt` and `log.txt`. The time interval is 1 hour, with $n = 3$ as the smoothing period and $k = 1.5$ as the sensitivity constant based on the data collected in the proposal section.

Code 2: ema.txt and log.txt example

```
1 [2025-07-06T20:00:03.621150+00:00] EMA:
  117.67 //ema.txt
2 [2025-07-06T20:00:03.625647+00:00] EMA:
  117.67, StdDev: 171.24, Threshold: 340.29
  //log.txt
```

Code 2 shows examples of `log.txt` and `ema.txt`. The `ema.txt` file stores the most recent EMA value, which is required to calculate the next EMA during the next execution. The EMA calculation process ensures continuity in the moving average process. The `log.txt` file records each threshold calculation, including the EMA, standard deviation, and final threshold. The `log.txt` file is useful for tracking threshold changes over time and for debugging or later analysis.

Code 3: Function to calculate EMA

```
1  def calculate_ema(data, previous_ema=None,
   alpha=None):
2      if not data:
3          return None
4      if alpha is None:
5          alpha = 2 / (N + 1)
6      ema_values = []
7      for i, x in enumerate(data):
8          if i == 0 and previous_ema is None:
9              ema = x
10         elif i == 0:
11             ema = alpha * x + (1 - alpha) *
               previous_ema
12         else:
13             ema = alpha * x + (1 - alpha) *
               ema_values[-1]
14         ema_values.append(ema)
15     return ema_values[-1] if ema_values else
       None
```

The code in Code 3 defines the function to calculate EMA. Line 1 specifies the function inputs, a list of log counts, a previous EMA value, and a smoothing factor α. Lines 2 to 5 check for empty data and assign α using the formula 2 when not provided. Line 6 initializes a list to store EMA values. The loop from lines 7 to 15 processes each log count. The first log count initializes the EMA. When a previous EMA value exists, the function calculates a new EMA based on it. Subsequent EMA values are calculated using the most recent EMA in the list. The function returns the latest EMA value. This function updates the alert threshold based on recent log trends.

5. Evaluation

Experiment Environment

The experiment uses EMA to dynamically set alert thresholds. When the warning log count stored in Elasticsearch exceeds the threshold, Elastalert automatically sends an alert to the CDSL monitoring team through the Slack channel.

- **VM Nodes:** 1 node (set on Mint ESXi)
- **Elasticsearch:** Version 8.13.4
- **Elastalert2:** Version 2.11.1
- **Python:** Version 3.12.3
- **Kibana:** Version 8.0.0
- **Slack Webhook:** Integrated with CDSL's Slack workspace

The system collects log data from devices and stores them in Elasticsearch. Kibana visualizes the data for manual monitoring. A Python script fetches warning log counts from Elasticsearch every hour. The script calculates the EMA and determines the threshold using the Equation (3). The value is then written into the Elastalert YAML rule file. When the log count exceeds the updated threshold, Elastalert sends an alert to the Slack channel.

Basic Experiment

The experiment compares the number of alerts sent by Elastalert when the `num_events` value is set manually by members of the CDSL monitoring team. The team is responsible for monitoring system logs based on a shift schedule. Each member provides a `num_events` value that they believe is most suitable for detecting warning-level logs. In addition, a separate YAML rule is used, where the threshold was set dynamically using EMA as proposed in this paper. All YAML files is executed at the same time. The test is run for 72 hours, from 00:00 AM, July 1, 2025, to 00:00 AM, July 4, 2025 (JST).

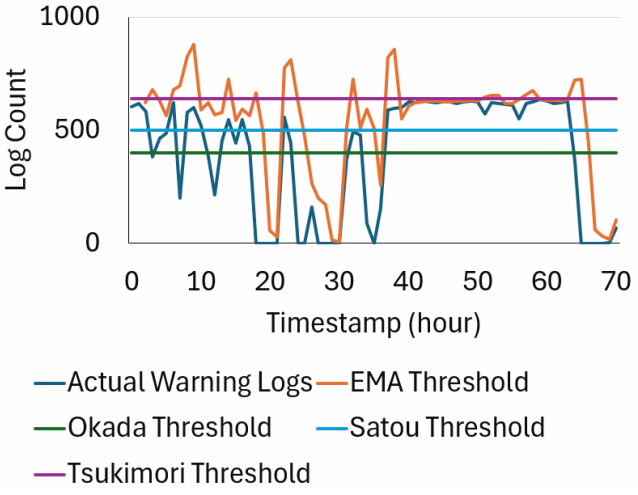

Figure 10: CDSL Members' Basic Experiment Graph

Figure 10 is the CDSL Members' Basic Experiment Graph. The graph shows the comparison between the warning log count and the thresholds set by each CDSL monitoring team member, as well as the dynamic threshold set by EMA. The x-axis shows the timestamp which is recorded hourly, and the y-axis shows the log count. The dark blue line shows the actual warning logs. The orange line shows the EMA threshold, which changes depending on the log activity. The other three horizontal lines show the Okada threshold, Satou threshold and Tsukimori threshold. These values do not change, even when the warning log count changes. The static threshold makes it less adaptive to fluctuations in the log data.

Table 4: CDSL Members' Basic Experiment Table

Name	**Threshold**	**Alerts Sent**		
		24h	**48h**	**72h**
Tsukimori	640	0	0	0
Satou	500	4	14	29
Okada	400	12	23	40
EMA	Dynamic	3	8	11

Table 4 shows the CDSL Members' Basic Experiment Table. The number of alerts sent by Elastalert for each threshold during the 72-hour experiment, visualised in Figure 10. Tsukimori set the highest threshold of 640, which resulted in 0 alerts throughout the test. Satou threshold of 500 resulted in 4 alerts in the first 24 hours, 14 alerts by 48 hours, and 29 alerts by the end of 72 hours. Okada threshold of 400 caused 12 alerts in the first 24 hours, 23 alerts by 48 hours, and 40 alerts by the end of 72 hours. These thresholds are constant, so they do not change even when the number of warning logs changes, making the threshold vulnerable to log spikes or drops.

The EMA-based threshold sent only 3 alerts in 24 hours, 8 alerts in 48 hours, and 11 alerts by the end of 72 hours. Even though the threshold changed over time, the number of alerts is the lowest. The table shows that the EMA threshold reduces false alerts while still reacting to the log data. It adapts to increases and decreases in log count, making the alerting system more flexible and accurate. The results proves that the EMA-based threshold is a good choice for dynamic alert rule management.

Evaluation Experiment

The evaluation experiment tests whether the EMA-based threshold setting improves alerting accuracy in a real environment. The alerting accuracy is calculated by dividing the number of true alerts by the total number of alerts sent during the monitoring period. The same is done for the missed accuracy, just the opposite. The formula is shown in Equation (4) and Equation (5).

$$\text{Alerting Accuracy} = \frac{\text{True Alerts}}{\text{Total Alerts}} \quad (4)$$

$$\text{Missed Accuracy} = \frac{\text{False Alerts}}{\text{Total Alerts}} \quad (5)$$

The value of n for the experiment is 3 and the value of k is 1.5. The experiment is done with the log.syslog.severity.name : "Warning" and host.hostname : lily logs. Lily is one of the EXSi servers set up in the CDSL Laboratory. Below is the flow of the experimental procedure.

(1) The Elastalert program is deployed together with the `ema.py` script.
(2) The `ema.py` script calculates a dynamic threshold based on recent log data retrieved from Elasticsearch.
(3) The script updates the `num_events` value in the Elastalert YAML rule file every hour.
(4) Elastalert uses the updated rule file to monitor logs and is executed continuously in CDSL's live monitoring environment.
(5) The experiment is conducted from 12:00 PM, July 7, 2025 to 12:00 PM, July 9, 2025 (JST), covering a 48-hour period.
(6) During the experiment, the number of alerts sent by Elastalert to CDSL's Slack channel is recorded.
(7) The system also stores threshold values and warning log counts in log files for analysis.
(8) A graph and a table are prepared to visualize the relationship between log activity and dynamic thresholds.

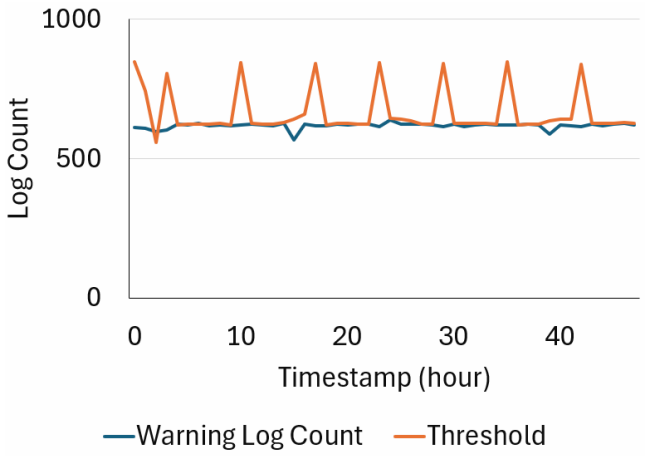

Figure 11: Evaluation Experiment Graph

The graph in Figure 11 shows the Evaluation Experiment Graph taken from from 12:00 PM, July 7, 2025 to 12:00 PM, July 9, 2025 (JST). The x-axis shows the timestamp. The y-axis shows the log count. The blue line is the log.syslog.severity.name : "Warning" Log Count. The orange line is the threshold set by EMA. The threshold is calculated based on the EMA and changes over time depending on the log activity. An alert is triggered whenever the log count passes the threshold.

The log values that pass the threshold are manually investigated using Kibana. Each alert timestamp is checked to confirm whether the alert corresponds to a real anomaly. When the log message is unrelated to system issues, the alert is counted as a false alert.

Code 4: SMART health warning

```
host.hostname:lily
log.syslog.severity.name:Warning
@timestamp:Jul 9, 2025 @ 11:52:48.324
event.original:<12>2025-07-09T02:52:48.324Z
lily smartd[2098743]: t10.NVMe_Samsung_SSD_
980_PRO_1TB_C0C6B811B8382500: REALLOCATED
SECTOR CT below threshold (0 < 90)
message:t10.NVMe_Samsung_SSD_
980_PRO_1TB_C0C6B811B8382500: REALLOCATED
SECTOR CT below threshold (0 < 90)
process.name:smartd summary:SMART health
warning: SSD may be degrading
```

- SMART health warning: SSD may be degrading
- NVMe I/O command failure: possible controller or disk issue
- VMkernel HPP module: NVMe command failure, device-level issue
- HPP throttle error: device returned error status (e.g., 0x281)

Code 4 shows the SMART health warning, which is one of the alerts that came out. The list shows all of the alerts which are common storage-related alerts, mainly involving NVMe SSDs. A SMART health warning suggests potential SSD degradation, while unclassified warnings require manual review. NVMe I/O and PSA command issues indicate controller or disk faults. VMkernel HPP-related alerts, such as throttle or command failures, point to lower-level device problems that impact system performance. Out of 11 alerts, 9 of the alerts had the above messages in the log data, which shows that it is a real alert.

Table 5: Summary of Alerts During 48-hour Monitoring Period

Alert Type	**Count**	**Percentage**
True Alert	9	81.8%
False Alert	2	18.2%

Table 5 summarizes the alerts generated during the 48-hour monitoring period. A total of 11 alerts were triggered, of which 9 were true alerts, accounting for approximately 81.8 %, and 2 were false alerts, accounting for approximately 18.2 %. The results show that the proposed method maintained a relatively low false alert rate, with most alerts being valid and actionable.

Analysis of Evaluation Experiment Results

The evaluation experiment shows that the EMA-based threshold detects warning log spikes while reducing the number of false alerts. During the 48-hour monitoring period, Elastalert sent a total of 11 alerts. Manual investigation using Kibana confirmed that 9 of these alerts were related to real system events, and 2 were false alerts.

The threshold values changed automatically based on the log activity. The threshold helped the alert system adjust to normal fluctuations in the log count. The graph shows that the EMA threshold followed the general trend of the log count and increased slightly during log spikes. The false alert rate was approximately 18.2%, calculated by dividing 2 false alerts by the total of 11 alerts. The result indicates that most alerts were accurate. In contrast to fixed thresholds, which are not dynamic, the dynamic EMA method produced a better balance between alert sensitivity and accuracy.

There were random spikes set automatically by the threshold based on the pattern of the threshold in Figure 11. The random spikes are due to when there is a small change in the value of warning log counts, the standard deviation value increases. The threshold is influenced by the value of the standard deviation, multiplied by the value of k which is 1.5, based on Equation (3). The change in values has caused the spikes inside the threshold graph.

The experiment demonstrates that the EMA-based method is effective for real-time alerting in live systems. The EMA threshold responds to log patterns and reduces unnecessary alerts. The low number of false alerts confirms that the method improves alert quality in practical use.

6. Discussion

When the warning log count remains constant for a long period of time, the standard deviation value becomes lower than the usual value. The lower standard deviation value reduces the value of the threshold in Equation (3). When the threshold becomes lower than the usual threshold, even a small increase in the log count exceeds the threshold and trigger an alert. The change creates unnecessary alerts that do not reflect real system problems. An algorithm to increase the threshold should be added to detect when the EMA and log count remain constant for a fixed period. When the static situation occurs, the algorithm should increase the threshold by a fixed amount to avoid false alerts caused by small changes in the log count.

The current implementation uses fixed values for the time interval, n, and k parameters. These values are hard coded in the `ema.py` script and are constant during runtime. When the pattern of the logs changes, the hard coded values become unsuitable. The system does not support automatic updates for these parameters. A Markov Decision Process (MDP) should be used to solve the problem. The MDP selects actions based on predefined policies. Each policy matches a different log pattern. The MDP chooses the best values for time interval, n, and k depending on the current log pattern. The MDP improves system flexibility and reduces the need for manual tuning [14, 15].

The syslog shows a clear daily pattern. The warning log count increases at specific times of day. There is a program made to shut down ESXi servers during midnight to save power. CDSL members also have other scheduled programs running on their personal ESXi virtual machines. The log count increases when these programs are active. A machine learning model such as Long Short-Term Memory (LSTM) should be used to detect and learn from these patterns. The LSTM model learns when the log count normally increases and adjusts the threshold accordingly. LSTM allows the system to increase or decrease the threshold automatically based on daily log activity [16].

7. Conclusion

This study proposes a dynamic threshold method for log monitoring using Exponential Moving Average (EMA) to reduce false alerts in Elastalert and improve accuracy. A Python script collects warning log data from Elasticsearch, calculates the EMA and standard deviation, and updates the threshold in the Elastalert YAML rule file every hour. The threshold is calculated using EMA, standard deviation, and a constant k. Various values of n and k were tested, with $n = 3$ and $k = 1.5$ producing the best results by reducing alerts while still responding to real spikes in the proposal section. The basic experiment compared fixed thresholds set by human members

with the dynamic EMA threshold. The EMA method sent 11 alerts which is less than the number of alerts sent by the fixed thresholds. and adapted better to log changes. The number of alerts sent was 11, 9 of which were confirmed to be real, and 2 false in the evaluation experiment. These results show that the EMA-based threshold reduces false alerts while maintaining accuracy. EMA improves Elastalert's performance by adapting to real-time log pattern and providing a more reliable alerting system for syslog monitoring.

References

[1] Gerhards, R.: The Syslog Protocol, RFC 5424 (2009).

[2] Virtanen, H.: Implementing Automated Log Based Alerts in a Patient Information System, Bachelor's thesis, Tampere University of Applied Sciences, Tampere, Finland (2017).

[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] Sonawane, P., Pingale, K. and Gawali, M.: Proactive montoring of server logs to prevent instant shutdown of the server using ElasticSearch, *INTERNATIONAL JOURNAL OF ENGINEERING DEVELOPMENT AND RESEARCH*, Vol. 6, No. 3, pp. 50–55 (2018).

[6] Chandola, V., Banerjee, A. and Kumar, V.: Anomaly detection: A survey, *ACM computing surveys (CSUR)*, Vol. 41, No. 3, pp. 1–58 (2009).

[7] 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).

[8] 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).

[9] Fikri, M., Herdjunanto, S. and Cahyadi, A.: On the Performance Similarity Between Exponential Moving Average and Discrete Linear Kalman Filter, *2019 Asia Pacific Conference on Research in Industrial and Systems Engineering (APCoRISE)*, pp. 1–5 (online), DOI: 10.1109/APCoRISE46197.2019.9318810 (2019).

[10] Oliner, A. J., Aiken, A. and Stearley, J.: Alert Detection in System Logs, *2008 Eighth IEEE International Conference on Data Mining*, pp. 959–964 (online), DOI: 10.1109/ICDM.2008.132 (2008).

[11] Hoblos, J.: Anomaly Detection on Univariate Time Series Data Using Exponentially Weighted Moving Average (AnEWMA), *Proceedings of the 10th International Conference on Internet of Things, Big Data and Security (IoTBDS)*, pp. 402–409 (online), DOI: 10.5220/0013437800003944 (2025).

[12] Zhou, Z.-G. and Tang, P.: Improving time series anomaly detection based on exponentially weighted moving average (EWMA) of season-trend model residuals, *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, (online), DOI: 10.1109/IGARSS.2016.7729885 (2016).

[13] Brown, R. G.: *Smoothing, Forecasting and Prediction of Discrete Time Series*, Prentice-Hall (1959).

[14] Smith, J. and Doe, J.: Toward a robust approach to multivariate time series anomaly detection, *Proceedings of the International Conference on Data Mining 2025* (2025).

[15] Kang, Y., Yin, B., Shang, W. and Xi, H.: Performance sensitivity analysis and optimization for a class of countable semi-Markov decision processes, *2011 9th World Congress on Intelligent Control and Automation*, pp. 799–804 (online), DOI: 10.1109/WCICA.2011.5970625 (2011).

[16] Prakash, S., Jalal, A. S. and Pathak, P.: Forecasting COVID-19 Pandemic using Prophet, LSTM, hybrid GRU-LSTM, CNN-LSTM, Bi-LSTM and Stacked-LSTM for India, *2023 6th International Conference on Information Systems and Computer Networks (ISCON)*, pp. 1–6 (online), DOI: 10.1109/ISCON57294.2023.10112065 (2023).

