Computer Networks 250 (2024) 110543

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Mitigating Alert Fatigue in Cloud Monitoring Systems: A Machine Learning Perspective

Fotios Voutsas [a], John Violos [b,*], Aris Leivadeas [b]

[a] Netdata Inc., 548 Market St #31942, San Francisco, 94104-5401, CA, USA
[b] École de Technologie Supérieure, 1100 Notre-Dame St W, Montreal, H3C1K3, Quebec, Canada

ARTICLE INFO

Keywords:
Cloud computing
Monitoring systems
Alert systems
Alert fatigue
Classification
Random forests

ABSTRACT

Next generation networks will be largely based on monitoring and telemetry tools that are essential for maintaining optimal performance, ensuring security, managing costs, and performing fault detection and resolution. An integral part of the overall monitoring strategy is alerting, which provides administrators with the necessary information to proactively or reactively manage and optimize network services. However, when monitoring systems generate an excessive number of alerts, many of which may not be actionable or may not represent critical issues, the phenomenon of alert fatigue occurs. Alert fatigue refers to a situation where the volume and the speed of the continuous influx of alerts becomes so overwhelming that the network administrators become desensitized and do not respond to them. To this end, and inspired by recent trends in network automation, where human intervention tends to be minimized, we introduce an alert fatigue mitigation mechanism in monitoring focusing on cloud computing infrastructures. In particular, a composite machine learning methodology is proposed in order to select which alerts will be hidden and which ones will be presented to the administrators. Additionally, to personalize the results, the proposed approach considers the level of users' experience along with the alert features to further optimize the accuracy of the alert filtering mechanism. The research has been conducted in a realistic environment of a leading monitoring enterprise, Netdata, which provided two datasets for testing our approach. Furthermore, the attained results of the filtering mechanism were evaluated by expert engineers of the company that verified the output of the proposed framework. Specifically, the outcomes confirm that our proposed methodology mitigates the alert fatigue problem with an accuracy that surpass 90% in most cases.

1. Introduction

Cloud computing has gained significant momentum over the last two decades. The majority of the daily services and applications used nowadays are offered through the various available cloud computing service models. Given the importance of this technology, several system and network administrators are responsible for providing a certain level of Quality of Service (QoS) to the cloud users [1], satisfy the availability, reliability and the recoverability of software platforms and hardware [2]. Hence, to ensure that the cloud platform and its applications perform as expected, the use of monitoring solutions is required [3]. Typically, software-driven tools are tasked with "monitoring" a distributed computing system and conveying pertinent information in an easily understandable format to administrators. This is done through the use of graph charts, pies, gauges, or just simple numeric variables.

Monitoring systems and tools, and especially the data that they generate, will be one of the core components of the next generation networks [4]. Specifically, these future networks, will be based on a collaborative decision making process between network administrators and artificial intelligence techniques, that will leverage telemetry data generated by monitoring probes that can be placed in critical points of presence in the infrastructure [5]. Through this collaboration the necessary network assurance could be achieved, while also reducing the need for constant human intervention [6].

An inherent part of monitoring systems is the alerting system. While alerts may manifest in diverse forms [7], they share certain common morphological characteristics. For instance, an alert must be distinguishable by a name and should include information about the system component triggering the alert, along with its status and the metric value responsible for the alert. In this context, two types of alerts are mostly under consideration. The first type refers to threshold-based raised alerts, indicating that a metric has surpassed a predefined threshold, accompanied by a "status" indicating whether the alert is

* Corresponding author.
E-mail address: ioannis.violos.1@ens.etsmtl.ca (J. Violos).

https://doi.org/10.1016/j.comnet.2024.110543
Received 29 January 2024; Received in revised form 28 April 2024; Accepted 24 May 2024
Available online 25 May 2024

critical or merely a warning. Alternatively, an alert may originate from an unsupervised machine learning model, which detects anomalous behavior and notifies administrators for corrective action or root cause analysis.

Accordingly, the alerting systems in large cloud infrastructures running many services may produce a significant amount of alerts of different priority and criticality, inundating administrators with vast amounts of information that can be proven challenging to manage [8]. This could cause a significant challenge called "alert fatigue". Specifically, alert fatigue occurs when an excessive volume of alerts desensitizes the network administrator who is responsible for addressing them, resulting in overlooked or dismissed alerts and delayed responses. The primary issue in monitoring cloud systems, lies in the sheer quantity of alerts. Responding to a single alert is manageable, even if it disrupts the working routine of an on-call administrator. However, handling a succession of a dozen alerts becomes more challenging, and as the number increases, there is a growing likelihood that the administrator may overlook something crucial.

The situation becomes even more difficult for new administrators of the monitoring tools who try to navigate through thousands of metrics and alerts. For instance, a normal software platform deployed on the premises of a cloud provider is expected to have around 4 000 metrics and around 80 – 100 different types of alerts. Additionally, a production system could have a number of metrics ranging from 10 000 to 20 000 and the alerts could be in the order of several hundreds. These figures reflect real data provided by our industrial partner organization, Netdata, that provides a flexible and modular real time monitoring and alerting tool [9]. Thus, it becomes clear that the administrator could be largely assisted by an automatic mechanism that can filter and prioritize critical events that could compromise the performance over less important alerts. The latter, motivated us to propose a novel alerting system, leveraging machine learning techniques to classify and filter the alerts that will end up to the administrators, with the ultimate goal to minimize the so called alert fatigue.

It is to be noted that Netdata have observed the phenomenon of alert fatigue also in experienced users.[1] In more detail, as an infrastructure is running its production workflows, alerts will be raised from the monitoring system, and in turn, presented to the system administrators. The more these administrators are exposed to alerts, the more likely it is for them to develop tolerance to those that are most commonly raised, normalizing their severity and ultimately ignoring them. This fact, that users with different levels of experience and different behavioral patterns suffer from alert fatigue, also motivated us to enhance the proposed machine learning methodology by including multiple binary classifiers corresponding to the multiple levels of user experience. It should be emphasized that our approach goes beyond of simply applying a machine learning model to a dataset to mitigate the alert fatigue problem. In contrast, we propose a composite and adaptive machine learning model that leverages the level of user experience, provides a feature selection technique among hundreds of available monitoring metrics, and is fortified by a human-assisted assessment. More precisely, the contributions of this paper can be summarized as follows:

- We present how a monitoring and alerting system works in a distributed computing environment and introduce the problem of alert fatigue in such a system.
- We propose and evaluate a lightweight alert filtering methodology that leverages the user experience to select only alerts that are important to each different level of user.
- We introduce and use two real datasets gathered by Netdata engineers from different time periods containing monitoring alerts of a real cloud environment.
- We conduct a human-assisted corroboration assessment, by soliciting the knowledge of expert engineers to assess the output of our results.

The rest of the paper is organized as follows. Section 2 highlights the related work regarding monitoring and alerting systems. Section 3 presents the main building blocks of how a monitoring and alerting system works. Section 4 proposes a machine learning methodology for alert selections based on the level of user experience in monitoring systems. Section 5 illustrates the results and the efficiency of the proposed methodology using two real datasets and one evaluation with human annotators. Finally, Section 6 concludes the paper giving some future directions in next-generation networks.

2. Related work & background

Our exploration of related work encompasses three key domains. In Section 2.1, we delve into the contemporary landscape of monitoring and alerting systems, explaining the imperative need for an alert filtering mechanism. Moving forward to Section 2.2, we scrutinize existing alert filtering techniques found in the pertinent literature, elucidating their constraints and limitations. Subsequently, in Section 2.3, we spotlight machine learning models poised to address these limitations effectively. Finally, in Section 2.4, we delineate our approach to mitigating the current monitoring and alerting systems limitations by employing an alert filtering mechanism that leverages a random forest model.

2.1. Monitoring & alerting systems

In the pertinent literature, monitoring systems are defined as applications that can provide awareness and observability over a given infrastructure of one or more processing and storage nodes [10]. They scale according to the infrastructure's size, meaning that they can monitor any number of machines and provide the desired metrics for each of them [10,11]. Such monitoring solutions can be either made in-house, or provided by dedicated providers to developers and system administrators. Each have their own set of features and limitations, due to design, or due to a paywall feature model [11]. Certain limitations in systems like these include the limitation of centralizing and streaming the time-series data and the amount of retention on each node. This retention concerns the time-frame that the data will be stored on each monitored node in order for a user to be able to go back in time and examine the time-series [7].

In order for such systems to be functional and effective, they have to be able to emit alerts towards the system administrator in case of certain incidents. This behavior is referred to as an "alerting system". Its existence in a monitoring solution is critical, since even when the data retention capability of the monitoring tool is unlimited and the data is centralized on a parent node, a human user cannot decode the vast amount of timeseries that are provided [12]. With an alerting system the limitation of retention can be managed, by summarizing time-frames into meaningful alerts. Such alerts are accompanied by a message with the health state of the node (either binary or with a string representation), and various attributes about the time-series that triggered the alert [13].

Monitoring systems are able to collect metrics from various different points in a system. For instance, in a medium-sized production infrastructure that we had access to there were 10 million metrics and from them came 200 pre-configured alerts that any of the nodes could raise as its own alert entry. These numbers are dynamic as the infrastructure evolves, and the users can even define their own alert entries with custom rules and thresholds. Thus, when a new node or individual

[1] Throughout this article, the terms "system administrator" and "user" are used interchangeably, reflecting the common scenario where the network administrator also functions as the user of a monitoring system.

Table 1
Netdata's alerting integrations.

Alerting integration	Subscription type
Email	free
Discord	free
Netdata Mobile App	paid
Amazon SNS	paid
Mattermost	paid
Microsoft Teams	paid
Opsgenie	paid
Pagerduty	paid
Rocket.Chat	paid
Slack	paid
Splunk	paid
Splunk VictorOps	paid
Telegram	paid
Webhook	paid

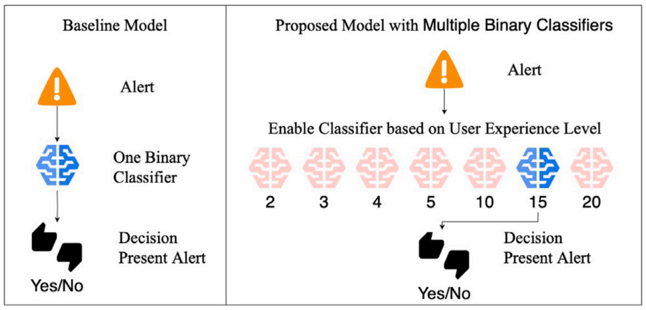

Fig. 1. Difference between traditional and proposed user-experience based alert filtering approach.

user policies are added to the infrastructure, alerts are produced. This behavior will then affect the inbox of the system admin responsible for reacting to alerts. Due to usually functioning with thresholds, all alerts can be in either "critical" or "warning" status, without any grading between a system-breaking alert or a normal time-sync alert on the clock of a node. In this case, the phenomenon of alert fatigue appears, which is known in cyber security [14] and even more in health care [15]. Hence, important alerts, which are blended with trivial and unimportant ones, can be missed inside an over-congested inbox. The term "inbox" represents the integration with which the user will receive the alert. For reference and as presented in Table 1, Netdata supports a host of alerting integrations, depending on the subscription type that the user has. Namely alert notifications are supported via: the Netdata mobile app, through Discord, Microsoft Teams, Slack, Splunk, Pagerduty and more. Furthermore, a manual grading on the alerts is not always available as it happens in cybersecurity [8], because each user has a different reaction pattern for a given alert.

2.2. Alert filtering

The alert fatigue problem from monitoring and alerting systems can be mitigated by creating a filtering mechanism. The concept of alert filtering has also been used in many different applications. For instance, email spam detection is a sector that can be greatly benefited from alert filtering mechanisms. In such a context, machine learning techniques can be used to leverage prior user reactions to alerts in order to filter new ones. As an example, the authors in [16] conducted research in the field of alert filtering and machine learning, by proposing the use of the Naive Bayes classifier, albeit in a smaller and less broad dataset. Additionally, alert filtering has been used along with machine learning in network intrusion detection systems [17], for filtering false positive alerts using a kernel density estimator. Lastly, there have also been applications in the field of software as a service that use text weighing techniques to extract information from textual alerts over the period of one day in order to filter security alerts [18].

In our previous preliminary work [19], we presented a methodology where a machine learning model is trained against the alert features and the reactions of users, and then provide an alert criticality prediction mechanism to fight the alert fatigue. From the experimental results of this previous work, we found out that the performance had low accuracy because inexperienced users' interactions were not separated from the experienced ones, resulting in poor predicting ability. The focal point of this setback is that users might have different reactions to the same alert and status due to workflow dissimilarity and overall experience with the tool. Furthermore, when training on a dataset of users' reactions, while they are already experiencing alert fatigue, the reaction patterns might have a big amount of noise in them.

To address the above limitations of our previous work and by identifying the lack of alert fatigue mitigation mechanisms in a cloud computing environment from the literature, we have reached out to Netdata to reinforce and enhance our preliminary findings. Specifically, based on extended discussions with Netdata we have learnt that experienced users have more stable and opinionated reactions to alerts while inexperienced users have more random reactions. For example, they might check every single alert or worse, not check any alert due to already experiencing alert fatigue. This made us to leverage the user experience level [20] to build a process in which we can dissect the learning workflow for groups of users on different experience levels.

2.3. ML & random forest

In different fields the issue of alert fatigue has been addressed with machine learning and specifically random forest models. For instance, such machine learning solutions have been applied to Clinical Decision Support Systems (CDSSs) where the aim is to minimize medication prescription errors, and to increase patient safety. As [21] presents, in these monitoring systems for patients, doctors get alerts about the effects of disease medication on patients, and react to them. Similarly to our case, doctors tend to under-react or have very hasty reaction patterns on an alert-fatigued inbox. Under-reacting means that they miss important alerts inside their congested –from unimportant alerts– program, and do not react to them, while the hasty reaction is recorded when the doctors react on repeated alerts without really paying attention to the alert details. The latter increases the chances of missing an important alert or a non-ordinary alert value [21,22]. The immediate and radical solution that some institutes chose was turning off the alerting system completely, as presented at [22]. Thus, for less radical solutions, random forest models and artificial neural networks among others were used to predict the reactions of the doctors to the alerts, with an aim to help the doctors with their responses on disease medication alerts from a CDSS [21].

2.4. Synopsis & beyond the related work

The synopsis from the literature is that the current monitor and alert filtering systems present significant drawbacks, while there is as a considerable lack of such systems in a cloud computing setting. Firstly, monitoring and alerting systems are inefficient, if the amount of monitored instances is vast and their behaviors are not identical. Secondly, basic alert filtering can help with alert fatigue, but does not provide optimal prediction performance due to user dissimilarity in experience and in workflow scenarios. For instance, all related works found use one classifier for all the alerts triggered by the monitoring system as depicted on the left of Fig. 1. As a result, the current alert filtering techniques are able to adapt to certain patterns that the monitored object has, but not to patterns that the alert responders follow based on their experience.

To overcome the above identified limitations, our work contributes to the field of network engineering by proposing a method that accurately presents relevant alerts to network administrators taking into

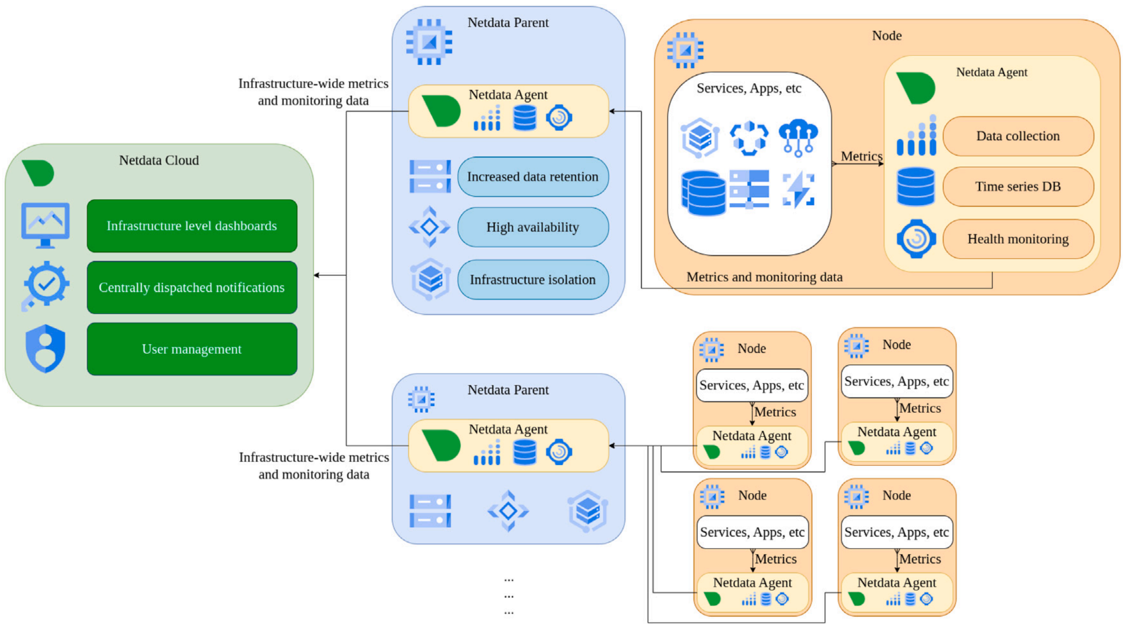

Fig. 2. Netdata architecture.

consideration their level of experience. To the best of our knowledge, this is the first work that adopts smart alert filtering mechanisms to the specific requirements of cloud monitoring systems. A significant contribution of our work lies in the introduction of a composite alerting model that incorporates multiple binary classifiers. When a new alert is triggered, it is directed to one of the binary classifiers based on the network administrator's level of experience. As illustrated on the right side of Fig. 1, our proposed model operates differently from other machine learning models explored in the related work. In more detail, our composite model learns from user experience and alert features, outperforming other examined alert filtering methods, as we will present in the subsequent sections.

3. Cloud monitoring & alert system

A cloud monitoring tool tracks, analyzes, and observes the performance, health, and parameters of cloud resources, aiming to offer real-time insights for administrators to identify issues, optimize resources, ensure security, and maintain system reliability, which are crucial for cloud infrastructure management. Hence, in this section, we present a typical architecture of a cloud monitoring tool following the Netdata paradigm. Subsequently, we focus on the alert system, which is the main topic of this work, and finally we position the alert fatigue notion within the cloud monitoring systems.

3.1. Architecture of a monitoring system

In this subsection, the architecture and main building blocks of a monitoring system are presented. It is to be noted, that the presented architecture is based on the Netdata tools used in the particular work, for reasons of consistency. Nonetheless, the same monitoring principles apply to other monitoring tools.

For instance, a monitoring system can be decomposed into two main functioning parts. The first part is normally installed on the node that we want to monitor, and has the role of collecting metrics of interest in the form of time-series data. That collection in most of the cases has to happen in a light (passive) way and also in some cases in an agnostic way too. For example, it may collect the count of queries in a database, but it should not read the query itself for privacy and security reasons. Another reason for this functioning part to be lightweight is to better support a wide variety of operating systems and working hardware.

The second functioning part is the visualization aspect of a monitoring system. It is responsible for presenting the collected metrics in a way that makes sense to a human user. In Netdata, this is accomplished by applying a classification over the time-series on their type and origin (e.g., metrics coming from a database, metrics coming from a network interface, etc.), on the topology (e.g., metrics grouped by the node from which they were collected, etc.) and many other groupings. This hierarchical classification is applied on the central metrics dashboard of Netdata and is the same for all the users. Also, as the metrics are time-series, many mathematical functions can be applied to them (i.e., mean, min, max etc.) to aid the needs of the user.

A high-level monitoring architecture of the Netdata tool is presented in Fig. 2 comprising of three main entities:

- **Node**. A node contains all the working services of a setup and it has a Netdata Agent that is responsible for:
  - Collecting data from various points of the node.
  - Retaining a time-series DB, for storing metrics locally and allowing the user to access them at any time.
  - Providing a Health monitoring mechanism that produces alerts that notify a particular user.
- **Netdata Parent**. A Netdata Parent is essentially another node that centralizes metric collection from its children. It provides:
  - Increased data retention, by storing the children's data for longer periods of time.
  - High availability by being a server used for centralizing the metrics, while also being available as long as no other workload is generated at this machine.
  - Infrastructure isolation by not allowing metrics to leave the infrastructure and to be stored in an outside system maintained by a third party.
- **Netdata Cloud**. A platform provided by Netdata that provides remote access to:
  - Infrastructure level dashboards, visualizing data from all nodes of an infrastructure in intuitive ways.
  - Centrally dispatched notifications, that can be identified by a certain "infrastructure ID" or so called "Netdata Space",

helping users know what went wrong and on which node/deployment.
– User management interfaces, allowing administrators to provide or limit access to certain dashboards and functions of the UI.

It is important to note that any Netdata Agent can be a Netdata Parent and vice versa. Additionally, all metric data are stored exclusively on premises, and Netdata Agents can delegate any of their functions (DB, Queries, Machine Learning, Health) to their Parents, to ease the load on that particular node.

3.2. Alert system

The primary goal of alerting, within the monitoring system, is to promptly notify administrators or relevant stakeholders about potential issues, anomalies, or critical situations that may impact the performance, security, or reliability of the cloud infrastructure. The key aspects of alerting in the monitoring of cloud infrastructures are provided below.

Threshold-based Alerts: Alerts are often triggered when monitored metrics (such as CPU usage, memory utilization, network latency, etc.) surpass predefined thresholds. These predefined rules stand as policies established by experienced developers to impose sensible default limits on the metrics associated with the alerts within the network. The alerts are then defined in script files, which can be further edited from the end-users in order to better fine-tune them for their own workflow. For instance, a script could implement a policy dictating that an alert should be triggered if the CPU usage surpasses a specified percentage or if response times exceed predefined acceptable limits.

Anomaly Detection: Some alerting systems utilize machine learning and anomaly detection algorithms to identify unusual patterns or deviations from normal behavior within the cloud infrastructure. This helps in detecting issues that may not be apparent through threshold-based approaches.

Event-driven Alerts: Some alerts can be triggered based on specific events or incidents, such as the failure of a server, a security breach, or the depletion of available storage space.

Critical Incident Notification: Alerts are prioritized based on severity levels, ensuring that critical incidents receive immediate attention. This allows administrators to focus on the most important issues first.

Multi-Channel Notification: Alerting systems typically support various notification channels, including email, SMS, instant messaging, and integration with collaboration tools. This ensures that administrators receive alerts through their preferred communication channels.

Escalation Policies: In case an initial alert is not acknowledged or addressed within a specified timeframe, escalation policies can be configured to notify additional personnel or teams. This ensures that critical issues are addressed even if the initially assigned personnel are unavailable.

Integration with Incident Management: Alerting is often integrated with incident management systems, facilitating a structured approach to incident resolution. This includes tracking, documenting, and analyzing incidents for continuous improvement.

Alerting plays a crucial role in maintaining the reliability, availability, and security of cloud infrastructures by enabling rapid response to potential issues. It is an integral part of the overall monitoring strategy, providing administrators with the information needed to proactively manage and optimize cloud-based services. An alert system, is a mechanism that many monitoring tools use in order to notify a certain user about the state of a node. Typically, they operate with “Statuses” in which the alert can be better described such as “warning”, “critical” etc. Such systems often report:

- The alert’s name
- The value that triggered the alert
- The current value of the metric that produced the alert
- The timestamp that the alert was triggered in a human readable form
- How long the alert has been in the reported status
- The identifier of the node that the alert belongs to
- The nature of the metric that raised the alert (e.g., Web Server, Database etc.)

All this information can then help the recipient to pinpoint what went wrong in their infrastructure and think of possible solutions. It is normal that all the metrics that are monitored from the software to be also accompanied by at least one alert, but the relationship between the two tends to be N alerts bound to 1 metric.

Furthermore, each alert operates with one or several threshold systems, that upon being exceeded triggers the alert into the status that the threshold specifies. There are also mechanisms in place to prevent small fluctuations of the metric around the threshold to trigger the alert. The interval that the alert is checked can normally be configured by the user, and its default value is dependent on the nature of the alert. Some alerts need to be checked every minute, for example alerts that are not mission-critical, while others that have utmost importance in an infrastructure, such as core temperature and free RAM for example, might need to be checked every single second. Once an alert is triggered, the administrator gets a notification from one of the available channels and can then run various helper functions to troubleshoot and take corrective actions.

3.3. Alert fatigue in cloud monitoring systems

As monitoring software scales to cover every possible metric that might provide valuable data to the users, so will the amount of alerts that will come with those metrics. As described before, this can cause an “alert fatigue” that can be triggered by the following reasons:

High Volume of Alerts: Cloud environments can generate a large number of alerts due to the sheer complexity and scale of the infrastructure. If these alerts are not effectively managed or filtered, the volume can become unmanageable.

Repetitive or Redundant Alerts: Continuous alerts about the same or similar issues without meaningful variations can lead to a sense of redundancy. Administrators may start to ignore or dismiss alerts, assuming they are not indicative of new or critical issues.

False Positives: If monitoring tools produce alerts that do not accurately represent actual problems or if they frequently trigger false alarms, it can erode trust in the alerting system. This can lead to administrators questioning the validity of alerts and potentially ignoring them.

Lack of Prioritization: When alerts are not appropriately prioritized based on the severity or impact of the issues they represent, it becomes challenging for administrators to discern which alerts require immediate attention.

Notification Overload: Alert fatigue can result from an excess of notifications through various channels (e.g., emails, messages, etc.), making it difficult for administrators to effectively prioritize and manage their response.

Ineffective Alerting Policies: Poorly defined or overly aggressive alerting policies can contribute to alert fatigue. Setting overly sensitive thresholds or generating alerts for non-critical events can overwhelm administrators.

As a simple example, an alert may be triggered when a node runs an ad-blocker and its blocker list has been expired 10 days ago. Let us assume that this type of alert has been pre-specified by default as “critical”. In this node, some time later, there is a Denial of Service attack being monitored and an alert about free system RAM is triggered, also in “critical” status. This may result in the following behavior. The administrator will receive two “critical” alerts, but their importance is nowhere near the same.

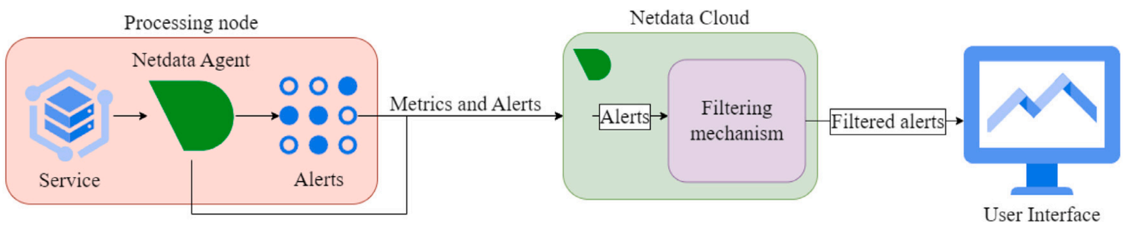

Fig. 3. Monitoring & alerting system.

If we extrapolate this example in an even small cloud infrastructure with few dozens of servers, which can host few thousands containers, each having its own metrics and alerts, the administrators could not know at every instant if the alerts they see on the screen are actually important or not. This creates a certain uncertainty if the administrator should click the alert and investigate it more or if they should flag it as a false positive and note to better configure that alert's threshold.

To mitigate alert fatigue, a filtering mechanism should be deployed, as shown in Fig. 3. In this way, only the important alerts are presented to the network administrator. Possible alert filtering strategies that may be followed are:

1. **Smart Alerting**: Implementing intelligent alerting mechanisms that prioritize critical issues and reduce the number of non-actionable alerts.
2. **Threshold Optimization**: Fine-tuning alert thresholds to ensure alerts are triggered only when there is a genuine concern, reducing false positives.
3. **Consolidation and Correlation**: Grouping related alerts or correlating events to provide a more holistic view, preventing the generation of redundant alerts.
4. **Regular Review and Adjustment**: Periodically reviewing and adjusting alerting policies based on the evolving nature of the cloud environment and organizational priorities.

Items 2, 3 and 4 come with many drawbacks compared to smart alerting, and are time consuming. Threshold optimization often happens inside configuration files and for the Netdata tool the configuration is on an 1:1 relationship with the alerts. Having hundreds of alerts would require a lot of man hours in order to properly configure them. Consolidation and Correlation in cloud environments that host various different applications, each with their own lifecycles in the system, cannot happen efficiently and holistically, while it also requires user input to have a sensible grouping. Regular Review and Adjustment is possible, but the main drawback is that it proposes editing the alert behaviors, which might in turn affect user reaction to the alerts. It also follows the previous pattern in which considerable man hours are needed in order to check and adjust the alerts, and as stated before, with the alert definition count being vast, a real infrastructure might be cumbersome to review in its entirety. These facts made us to design a machine learning perspective that can enable a smart alerting strategy.

4. Proposed methodology

Our proposed methodology resolves the problem of alert fatigue by using a filtering mechanism that provides the user with a score for each alert, in the form of a probability. The decision of the user to click an alert or not declares the user reaction. In order to build this filtering mechanism, we follow a supervised machine learning approach where the training is based on the alert features and the reactions of the users and the inference provides the probability of an alert to be clicked. Our methodology includes a group of models, where each one of them corresponds to the user's experience in monitoring and alert systems. Specifically, our methodology aims to leverage the fact that new and less experienced users often tend to have a more random behavior when selecting which alerts to see, in contrast intermediate users have more guided patterns in their reactions, while expert users have even more specific patterns. Thus, we have proposed a composite model that utilizes many sub-models to satisfy all the different user groups we define. In the following subsections, we provide the details of each component of the proposed solution.

4.1. Training stage

At the training stage, the model first receives a dataset consisting of alert features such as metric value, alert status, metric classification, and family of alerts. These features describe the system's state for a given alert domain, and reactions that indicate the users' interest to open these alerts. Then, the data gets preprocessed and forms sub-datasets, directly linked with the above-mentioned user groups. Each of them are represented with different feature vectors. New users will have fewer alert reactions and consequently less features to represent them than experts. The historical user interactions with the alerting system are vectorized and grouped in the sub-datasets in order to train the corresponding random forest models.

4.1.1. App interaction and creation of the dataset

The Netdata Console monitors the cloud infrastructures and produces alerts to the owners of the infrastructure as shown in the first component of Fig. 4. The users will take then the decision to open and consult the notifications or not. This behavior gets monitored and logged in an internal database that is then ingested by the proposed model.

4.1.2. Preprocessing

The preprocessing stage, illustrated in the second component of Fig. 4, begins once the training dataset has been generated. Initially, using feature importance ranking measure [23], a subset of features is selected from the dataset in order to be processed. Then the data features that are of string type are factorized, in order to use them as distinct values and to be able to interpret them as numbers inside the model. The purpose is to reform the data into several feature vectors, which provide a personalized aspect to each entry, giving past reactions of the user on the specific alert. New users will not have a broad history and will form smaller vectors, while experienced users will form larger vectors. Moreover, the timestamp and user_id features inside each alert entry are used, in order to group them per user and sort them by time. This pivot is vital for the Feature Vectorization mechanism and its subcomponents, as it will be later described, along with the Overflowing and Backfilling mechanisms used during the preprocessing.

Overflowing mechanism. The overflowing mechanism is responsible for enriching potentially sparse data of the training sub-datasets with data that might not be directly applicable to each of them. With the use of this mechanism, an entry that fits in one feature vector sub-dataset, can be truncated to fit into the rest of the smaller feature vector sub-datasets, contributing this way to their final data size. This approach is safe to do, since as the size of the sub-datasets increases, so does the experience of the users. In this case, there is no risk of "polluting" a sub-dataset with data from inexperienced users.

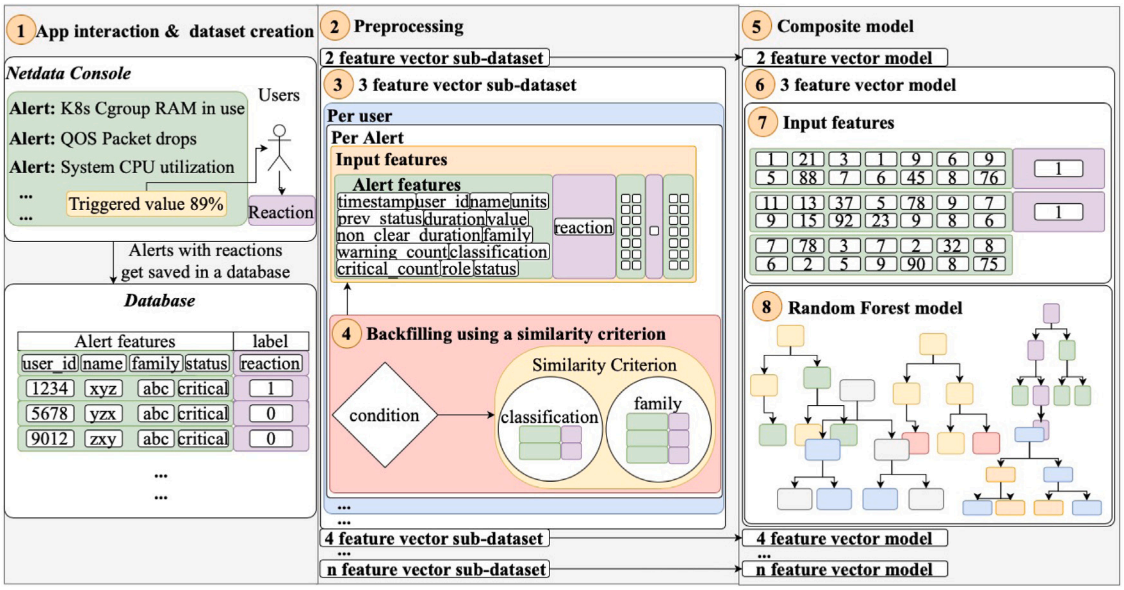
1 App interaction & dataset creation
Netdata Console
Alert: K8s Cgroup RAM in use
Alert: QOS Packet drops
Alert: System CPU utilization
Triggered value 89%
Users
Reaction
Alerts with reactions get saved in a database
Database
Alert features
label
user_id name family status
reaction
2 Preprocessing
2 feature vector sub-dataset
3 3 feature vector sub-dataset
Per user
Per Alert
Input features
Alert features
timestamp user_id name units
prev_status duration value
non_clear_duration family
warning_count classification
critical_count role status
reaction
4 Backfilling using a similarity criterion
Similarity Criterion
condition
classification
family
4 feature vector sub-dataset
n feature vector sub-dataset
5 Composite model
2 feature vector model
6 3 feature vector model
7 Input features
8 Random Forest model
4 feature vector model
n feature vector model

Fig. 4. Training stage.

Feature vectorization mechanism. This mechanism (component 3 in Fig. 4) is responsible for creating different sub-datasets that accept feature vectors of data that are personalized and are fed into the model for training. Each feature vector type has a numbered prefix indicating its feature representation. The ith feature vector type will have $i-1$ complete alert entries (features plus their label), and the features of the alert that is up for prediction. So the 3 feature vector type will have three alert entries and two reaction labels. The amount of sub-datasets we create is based on differentiating the users' experience levels into many grades. For instance, a feature vector of 2 alerts will have: (i) an alert's feature-set along with the user's reaction to that alert and (ii) an alert's feature-set without a reaction.

That final missing reaction is the label that the model will give a prediction for. With this process, the data are grouped so that the model does not train on just one interaction per entry, but on a feature vector of data. For instance, in the third component of Fig. 4, a user has two interactions on a particular alert, and his third reaction can be predicted by taking into consideration the two previous reactions and every individual alert's features.

The filling of these sub-datasets is performed for every status of every alert the user has reacted upon. If an alert appears first as critical and then as a warning, then there will be two separate entries in a given sub-dataset for a user. At the end of the preprocessing there will be n amount of sub-datasets, each populated with a user experience level, going to be inputted into one of the respective binary classifiers of the composite model.

Backfilling using a similarity function. The backfilling mechanism works by using a similarity function in order to not ignore users that do not meet the feature vector type data criteria for a sub-dataset. It is presented in the component 4 of Fig. 4. A user might only have two interactions while the sub-dataset might be requesting for three. In that case, a hard-coded similarity criterion is used to be able to fill the rest of the required data with context-similar reactions from the user on other alerts.

By using this mechanism, we ensure to empower some sub-datasets that might not have a lot of data to train on, due to the origin of the dataset or due to the n features of the feature vector at hand. The particular challenge can be modeled as a neighborhood problem. In a human analogy, if we cannot find the owner of a house, we will knock on the neighbor's door, and if no one is there either, we will go knock on a wider range of neighborhood. Similarly, for our problem at hand, when a user does not have enough alert reactions for a sub-dataset, our framework will search at that alert's closest neighbors, which alerts belong to the same family. For example, all CPU related alerts have the flag of "CPU" family, network interface related ones have "Network" and so on. Thus, if few entries are found but the feature size of the vector is still not met, we will try to look at a wider neighborhood, searching for alerts of the same classification as the alert under consideration.

4.1.3. Composite model

The purpose of utilizing a composite model instead of applying a simple machine learning model, is to be able to give a prediction for each user scenario, while also leveraging the amount of user's reactions. Applying a simple model, would mean that it should be trained on all the alert entries from the whole dataset unconditionally, and would not be able to adapt to all the different levels of user experience. In contrast, taking into consideration the level of user experience, their previous selections and the alert features, the proposed composite model can give a prediction which a common model cannot provide.

The composite model works by accepting each sub-dataset of data coming from the preprocessing stage, and provides a binary classifier for each. This procedure is illustrated in the fifth component of Fig. 4. Each binary classifier is trained based on the data of the corresponding feature vector of n features. Random Forest models are used for every binary classifier and are being trained against the input data. The amount of sub-models can vary as a different amount of sub-datasets means that there can be any number of binary classifiers, while the training set can also have different lengths in the dataset depending on the numbers of features of each feature vector.

4.1.4. Random forests

We chose random forests [24] as our binary classification model for the alert filtering task. Random forests belong to the ensemble family of classification models. Such methods advocate using multiple models instead of just one to achieve greater accuracy and minimize overfitting. This strategy is particularly suitable for our situation, where diverse behaviors of users are more accurately predicted using these methods. Random forests work by generating multiple decision trees during training. They utilize a technique known as "bagging" to form various training datasets from a single sample dataset, to then input them into different trees. Afterwards, each tree makes a prediction, and the most common prediction among the forest becomes the model's final decision.

Table 2
Features that describe an alert.

Feature name	Description
timestamp	The processed timestamp for the alert.
user_id	A unique identifier for each user.
name	The alert's name (e.g., "CPU usage"), which is a string containing the name of the alert in the notification.
family	The alert's family (e.g., "CPU" family of alerts), which refers to the category/group of alerts that a particular alert belongs to.
prev_status	The alert's previous status, meaning the state ("critical", "warning" or "clear") that the alert was before it got raised in the current state.
duration	The alert's duration (e.g., 15 min, in second format), which signifies how long this alert has been in the reported state.
non_clear_duration	The alert's non-clear duration (e.g., 5 min, in second format), which covers the case where an alert in its previous state was not clear, for example an alert first gets raised as "warning", and then gets escalated to "critical", this value would be the total duration that the alert is raised.
role	The user's role, for example "sysadmin", "dbmaster" etc.
status	The alert's status, which can be any from "critical", "warning" or "clear".
value	The alert's value (e.g. "99"), which is typically an integer value, indicating the alert's value, that will be accompanied by a unit.
warning_count	The infrastructure's total count of "warning" alerts (e.g., "3" warning alerts in total).
critical_count	The infrastructure's total count of "critical" alerts (e.g., "7" critical alerts in total).
classification	The alert's classification (e.g., "Utilization, "Latency", Error" etc.), which refers to the nature of the alert.
units	The alert's units (e.g. "%", "Errors", "Mbits/s") indicating the units that accompany the value of the raised alert.
Reaction	A Boolean value indicating whether or not the user clicked the notification, "1" clicked, "0" not clicked.

Table 3
The seven models included in the composite model.

Model	Model description based on the number of users alerts
Model-2	Beginner users with two interactions with the alerting system for a given alert
Model-3	Beginner users with three interactions with the alerting system for a given alert
Model-4	Intermediate users with four interactions with the alerting system for a given alert
Model-5	Intermediate users with five to nine interactions with the alerting system for a given alert
Model-10	Intermediate users with ten to fourteen interactions with the alerting system for a given alert
Model-15	Experienced users with fifteen to nineteen interactions with the alerting system for a given alert
Model-20	Experienced users with more than twenty interactions with the alerting system for a given alert

4.1.5. *User experience & alert features*

We employ random forest models for inference, trained on datasets tailored to users' experience levels. These models process alert features and historical user responses to generate precise outputs. This differentiation is crucial, as what may constitute a critical alert for one user could be considered a routine stressor for another. NetData can monitor hundreds types of metrics, all of which could serve as potential features for an alert filtering mechanism. Following discussions with NetData engineers, we identified the 150 most relevant metrics. Subsequently, by utilizing Feature importance [25] with a forest of trees, we determined the 13 key features outlined in Table 2, while the last feature "Reaction" refers to the labels of the dataset. The users' experience level is quantified based on their historical engagement, specifically the number of interactions they have had with alerts via clicks within the monitoring tool, as presented in Table 3.

The different feature vectors of data that are created ensure that all the levels of experience with individual alerts are captured. As the experience of a user increases so does the stability in the type of alerts they click, depending on values, statuses, classification and more. This enables the composite model to be trained differently per experience level. Thus, on an alert that has a "warning" status but the users' reactions render it unimportant, the model will be expected to give a low probability to click, to indicate that the user does not normally click this particular alert with the features at hand. The prediction needs to be in a form of recommendation, so that the user is not funneled or instructed to a specific behavior.

4.2. *Inference stage*

The inference stage consists of the steps required to produce a prediction for a certain sample of data. It is presented on Fig. 5, with the preprocessing component being the same as in Fig. 4. Similarly to the training stage, when the monitoring and alerting system outputs a potential alert, the alert's features get sent to the composite model (component 1). The data gets inputted and prepossessed by the preprocessing component (component 2). Finally, the composite model ingests the feature vector and inputs it in the right binary classifier. A prediction is then produced, giving a binary result for the alert (components 3–7).

This result can also be visualized back to the user as a percentage of the importance of the alert, so it is better understood. Such a visualization can be found in the component 7 of Fig. 5. As an example, if an alert emerges and the model predicts that the user should click the notification with a high but not an absolute percentage, the prediction will be rendered, for the sake of the example, as "86% probability" that this alert should be of interest. Obviously, this is a quantitative example, in real next generation networks and following trends of network automation more qualitative values could be used as "high",

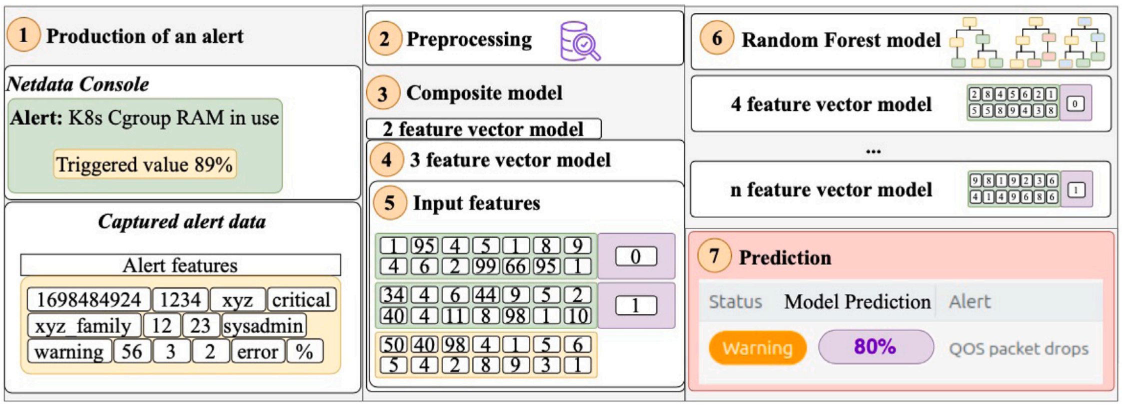

Fig. 5. Inference stage.

Table 4
Comparison among Different Machine Learning models.

	Accuracy	Precision	Recall	NPV	Specificity	Fall-out	F1-Score
DNN	0.424	0.424	1	0	0	1	0.596
Random forest	0.706	0.732	0.771	0.667	0.617	0.382	0.751
Decision tree	0.669	0.720	0.694	0.605	0.634	0.365	0.707
Naive Bayes	0.552	0.664	0.449	0.481	0.692	0.307	0.536
Logistic Regression	0.578	0.587	0.895	0.513	0.149	0.850	0.709

"medium", etc. [6], prompting the user to act upon only in very critical scenarios.

5. Experimental evaluation

5.1. Experimental setup

To experimentally evaluate and compare our proposed methodology, two comprehensive datasets were gathered in collaboration with Netdata, while one human-assisted evaluation took place by network experts. The first dataset was made through an internal log of the alert notifications sent to users over the course of September 2022 to May 2023, referred to as the ND2022 dataset. The second dataset was made from interactions after May 2023, referred to as ND2023 dataset. The ND2023 dataset would be used for further experimenting with the architecture of the composite model and for evaluation purposes. More specifically, we tried training the composite model with the ND2022 dataset and then testing it against the ND2023 dataset. In addition, during the gathering of ND2023 various major updates in the user interface and the alert presentation were made in the Netdata interface. The use of a different Netdata user interface in the constructing of ND2023 dataset compared to the ND2022 dataset adds value to the evaluation of the generality of our proposed methodology.

The datasets were organized to include a binary "click" label indicating whether the administrator clicked on the alert notification or not. This facilitates the prediction of administrator behavior based on the features embedded in the alert. Consequently, it becomes possible to model how frequently administrators with similar features and system states respond to a given alert. As a result, there is a conceptualization of integrating a new element into the alerting system, capable of assessing the significance of an alert by analyzing how a user of certain experience typically responds to such alerts.

The ND2022 dataset had a length of 150,000 entries with a size of $411MB$. It was generated from 13,072 individual real users and had an even split between the two target labels. The ND2023 dataset had a length of 75,000 with a size of $206MB$, generated by 7819 unique users with the same label split characteristics. The two labels are the label-0 which represents not-clicked alerts and the label-1 which represents the clicked alerts. In those datasets we had equal amounts of label-0 and label-1 instances in order to not bias the model in a certain action, as the outcome of the interaction on an alert is based heavily on the preferences and the experience of the user.

Since the alert notification is one of the main functionalities in a monitoring and alerting system we want to have an accurate filtering on the alerts for the different types of users. Through a series of experiments and by trying out various approaches in terms of alerts grouping, users grouping, and personalization methods, we are able to create a methodology that generalizes very well. While experimenting, we tried to test different personalization techniques, such as trying to cluster the input data into groups and then making a model per cluster. That proved to be inefficient and it was better to have only one model and to not dissect the users into clusters. The main cause was that some clusters were having too few data compared to others, making an inferior model in terms of prediction performance. Another personalization technique we tried was to group the data and make models for different alerts, but that proved to not be optimal, as different users do not react in a similar way to alerts. User A might click the alert because it is vital for his setup, while User B might not click the alert as it is an expected alert in his pipeline.

To further evaluate the performance, we utilized human experts to annotate a dataset that would then be tested on the composite model. Essentially, a human engineer would react to an array of alert entries, as if they were in a realistic scenario. With this behavior they would be keen on clicking some alerts, while not so interested on looking at others. Then a dataset would be built, and it would be inputted into the trained composite model for a prediction to be made.

5.2. Evaluation metrics

In order to evaluate and compare the performance of our proposed model we should firstly discern the positive from negative predictions and whether the prediction is true or positive. Specifically, positive predictions refer to alerts that are clicked by the users, while negative predictions refer to alerts ignored by the users. True positive or (tp) refers to how many predictions the model classified as 0 with them actually being 0. False positive (fp) refers to how many predictions were classified as 0 but in reality they were belonging to the 1 label. True negative (tn) refers to how many predictions the model classified as 1 with them actually being 1 and finally false negative (fn) refers

Table 5
Evaluation and comparison between the baseline and composite model with two datasets and human evaluation.

Performance metrics	Baseline model ND2022 Train & Test	Composite model with only name alert ND2022 Train & Test	Composite model ND2022 Train & Test	Composite model ND2022 Train & ND2023 Test	Human evaluation
Accuracy	0.706	0.756	0.922	0.939	0.826
Precision/PPV	0.732	0.614	0.838	0.953	0.826
Recall	0.771	0.614	0.935	0.886	0.843
NPV	0.667	0.822	0.968	0.931	0.826
Specificity	0.617	0.822	0.916	0.972	0.808
Fall-out/FAR	0.382	0.177	0.083	0.027	0.190
F1-Score	0.751	0.614	0.884	0.918	0.830

to how many predictions were classified as 1 but in reality they were of 0 label.

The metrics applied in the testing part of the dataset are the following [26]:

- Accuracy, $(tp + tn)/(tp + tn + fp + fn)$, which is the percentage of correct predictions that a model was able to achieve.
- Precision, $tp/(tp + fp)$, which presents the proportion of positive predictions that were actually positive.
- Recall, $tp/(tp + fn)$, which shows what proportion of real positives was actually classified as positive.
- Negative Predictive Value (NPV) $tn/(tn + fn)$, which presents the proportion of negative predictions that were actually negative.
- Specificity $tn/(tn + fp)$, which is the proportion of true negatives that were correctly predicted.
- F1-score $(tp)/(tp + (1/2 * (fp + fn)))$, which is defined as the harmonic mean of precision and recall.
- Fall-out $fp/(tn + fp)$, which is the proportion of negative predictions incorrectly identified as positives.

5.3. Outcomes

The composite model utilizes seven sub-datasets of different feature vectors in order to train the seven different models presented in Table 3. Each model represents a different level of user experience in the alerting system based on the number of clicks on the alert notifications. The sequence of these models was the most inclusive in terms of users being eligible to get a prediction, while also being the most accurate among multiple combinations that were tried. Table 4 presents the comparison among various machine learning models using the ND2022 dataset. The goal of this evaluation is to corroborate our selection of random forest against other available ML algorithms and to justify its integration in the proposed composite model. The results show that indeed the random forest model is the most accurate and the most appropriate for our problem at hand.

Our proposed methodology is also compared against a generic machine learning filtering method and a prediction method that is based only on the alert names and the levels of user experience. We call the former as baseline model and the latter as composite model with alert name. The baseline model relies on one simple random forest as proposed in our previous work [19] and it does not utilize any user grouping or feature vectorization techniques. In addition, this model does not leverage the levels of user experience. The baseline model is able to score a 75.1% in the F1 metric and 70.6% in the accuracy metric as can be observed in the second column of Table 5. Using a method that takes only the alert name as input feature and also the reactions of the users for every level of experience a 61.4% F1-score and 75.6% accuracy can be attained as noticed in the third column of Table 5.

Next, the proposed composite model is evaluated and its results are summarized on the fourth column of Table 5. As can be noticed, the performance is considerably high using the ND2022 dataset for testing and the particular outcomes confirm the applicability of our proposed composite model. The F1-score metric lies at 88.4%, which is a 17.33% improvement from the baseline model. Regarding the computational overhead of the filtering mechanism in the Netdata tool, we made multiple tests and measured that the average response time was close to 35 msec for a batch of one hundred alerts, which renders our framework practically as a real-time model.

For the next set of experiments, the generalization of our approach was evaluated, when testing in the ND2023 dataset. For this reason, the composite model was first trained with the entirety of the ND2022 dataset, and then tested with the ND2023 dataset. The outcomes are summarized in the fifth column of Table 5. The allocation of data was a 70%–30% split in terms of entries between the datasets. From this experiment, we ensure that our model is not being overfitted over the ND2022 dataset. Additionally, we want to show if our model is performing well on a dataset where there are time periods that new users are coming in, while old ones are becoming more experienced and many user interface changes were made. Since many users moved from the beginners group to more experienced ones, the performance of the F1-score metric was increased at 91.8% and the accuracy at 93.9%.

Furthermore, the evaluation also focus on how each separate binary classifier included in the composite model performs. For the particular evaluation the composite model was trained on ND2022 and tested on ND2023. Table 6 summarizes the evaluation metrics. As illustrated, the accuracy is higher than 90% in all binary classifiers and also there is an observed pattern where the accuracy tends to increase with the rise in user experience level. This outcome is expected, since the models benefit from more historical data in each feature vector length increment and because of the fact that as the users become more experienced in the monitoring tool the behavior in selection of the alerts becomes more predictable.

In order to compare the Random Forest architecture with other machine learning methods, we did the same experiment on ND2023 using a Long Short-Term Memory layer model. The results were 0.910 in the F1-score metric, a slight decrease from the Random Forests, but the major difference was found in the response time, with the model taking 400 msec, more than 1000% increase from the architecture using the Random Forests. Thus, such results prove our method is better in terms of prediction ability, but most importantly in terms of model response time.

Finally, we evaluated the performance of the model with human annotators that used Netdata and the proposed personalized filtering system. The human annotators had a specific reaction profile. They would react on specific alerts that would correspond to a certain workflow scenario, and not react to others that they were not important for their workflow (even if the alerts were critical). This was done to simulate various roles of monitoring users, like a Web Server Administrator, a Database Administrator and a Network Administrator. The total amount of human annotated reactions was 200 alerts. These alerts were pooled out of real alert samples in a random way, in order to give to the annotators alerts of all kinds. The goal was to use the trained composite model to predict the annotators' very specific and demanding behavior, to then observe the usability of the model. As an example, Network Administrators would not click on alerts related to a database or a virtualization technology. They would only react on alerts based around the infrastructure's network setup. The outcomes of this evaluation are presented in the sixth column of Table 5. The F1-score

Table 6
Evaluation and comparison for each model of composite model (training with ND2022 & testing with ND2023)

Binary classifiers	Accuracy	Precision	Recall	F1-Score
2	0.929	0.945	0.909	0.930
3	0.904	0.945	0.780	0.855
4	0.985	1.0	0.918	0.957
5	0.968	0.944	0.809	0.871
10	0.991	1.0	0.928	0.962
15	0.968	1.0	0.875	0.933
20	0.974	0.956	0.977	0.967

metric was 83.0% and the accuracy was 82.6%, two very good results on an evaluation test with very specific user scenarios.

It should be noted that the performance of the experiment with the human annotators had an expected drop in performance, when compared to the previous experiments of Table 5. Firstly, it is important to note that the annotators were initially excluded from the training datasets, as they were internal engineers, and we chose to not include them in the sampled datasets ND2022 and ND2023, thus their personal behavioral patterns were not used for training. This decision was made in order to not pollute the dataset with reactions from developers of the software, that might have testing environments producing a lot of alerts, or themselves behaving in a testing manner. Secondly, the engineers that used the alerting system along with the composite model, were aware of the situation and gave a certain amount of attention to all alerts that they were required to annotate. Thus, their behavioral patterns were different from those of a user that operates a monitoring system in his daily workflow, leading to this accuracy discrepancy.

5.4. Discussion

Commercial monitoring and alerting systems do not provide more personalized information like user locations, applications types or website behavior data. Therefore, we cannot use well-established personalization methods and we concluded that the most appropriate information to personalize and improve the performance of the alert filtering is the level of user experience. The investigation on the historical monitoring data showed that the user experience can be measured better by the number of clicks on alerts than any date/chronological information.

In terms of applying our framework to the monitoring tool, the results of the model can be visualized in the user interface of the Netdata tool, in two major ways. Firstly, alerts with predictions that have very low probability to get clicked can be hidden behind a button with a counter that indicates only their population count. In contrast, the high probability alerts can be presented in a sorted manner according to the probability that a particular user would have clicked them. Different color codes according to the severity of the alert can also be used for better reaction and sorting from the user.

Regarding the selection of the random forest models, this was justified from the results obtained in Table 4. In more details, this table presents the outcomes from various machine learning models with a simpler model architecture using the same dataset for training and testing as our composite model. Deep learning models, based on a feed-forward model, were not able to distinguish the binary prediction (Specificity was 0%) thus being able to only achieve 42% accuracy. The Naive Bayes classifier and the Logistic Regression models were performing on a very average score, with an accuracy of less than 60% accuracy. The decision tree model was the closest in performance, but was lacking against the random forest model in almost all metrics.

Additionally, we should take into consideration that an alerting and monitoring system should not add considerable overhead to the infrastructure. In edge and cloud computing environments the processing resources are valuable and they should be mainly used by the real applications hosted in the infrastructure, minimizing their utilization by the management and orchestration services. This is the reason that we avoided complicated deep learning models and we resorted to classic machine methods such as random forests. The experimental comparison in terms of response time confirms our assumptions, as evidenced by the average inference time for a batch of one hundred alerts being 35 ms for our approach and 400 ms for the LSTM approach.

With this as a starting measure, we decided to use the random forest model in order to build a more complete architecture for the problem at hand. Such an architecture can be leveraged by a smart alerting system and help new users that get flooded with alerts, enabling them to focus on the important tasks and alerts instead of those that can adversely impact their concentration and cause an alert fatigue. This is crucial since especially new users can get easily confused when facing an enormous number of alert messages, which can discourage them from continuing the learning process and the use of the monitoring tool. Additionally, it can also increase the productivity of more experienced users by indicating if an alert at hand should be checked or not. The model would be essentially capable of detecting false positives in the monitored infrastructure, making it easier for system administrators to know what the state of their deployment is.

The above discussion and the applicability of our proposed model are confirmed experimentally by the second set of experiments performed. In particular, as shown in columns 4 and 5 of Table 5 the evaluation metrics of the composite model have been improved in all aspects compared to column 2, which is the baseline model. Specifically, the accuracy metric that indicates the overall ability of the model to predict correctly has increased by 22.3% from the baseline model. Furthermore, the model has very good precision score or PPV, meaning that it can classify well the class of the alert being clicked (true positive). A high Recall score also means that the model is able to not incorrectly classify samples as positives with a ratio of 88.6%. In contrast to Precision or PPV, Negative predictive value, NPV is also high, meaning that the model can classify well on the class of the alert not being clicked (true negative). Specificity can be explained as the true negative rate. For instance, the closest it is to 1, the better the model's ability is on predicting negative labeled samples. Finally, a low Fall-out score indicates that the model is capable to not mistakenly classify negative-labeled samples as positive class.

It is also worth mentioning the accuracy improvement achieved in the 5th column of Table 5 compared to the 4th column. The accuracy achieved when using the ND2022 dataset for both training and testing is lower than training the model on the entire ND2022 dataset and then using the ND2023 dataset as a test set. This embraces the personalization definition in this paper, since in the ND2023 dataset the users are more experienced both by using the software, but also from the user interface updates that enhanced their experience when dealing with alerts. Therefore, the less experienced users on the ND2022 dataset have a degree of unpredictability in their behavior, that is lessened on the ND2023 dataset. Experience, and its product, predictability are the main reasons the model benefits on performance with the ND2023 dataset, as there is more history to the reactions of the users but also the behavior of the users themselves is more predictable due to their increased experience.

In addition, the performance achieved in the 5th column of Table 5 can be justified by the fact that the users became more experienced over the coarse of time. This shows, that the composite model did generalize well on these new samples and users. Users having more experience means that their behavior is stable and has sense, in contrast with the inexperienced users that have mostly random and unpredictable behavior. Furthermore, the feature set of Table 2 remains a good selection to represent the state of the alerts for both datasets. In contrast when we tried different other feature combination (i.e., by incrementally adding features for training) there was a significant drop in the performance. As an example, the third column of Table 5 shows the drop in the accuracy when the same methodology is kept but only the name of the alert is used as feature. Similar evaluation outcomes were observed for different combinations of features.

Another interesting topic of discussion arises from the findings of the sixth column of Table 5. In the particular experiment, the model was trained with the ND2022 dataset considering alerts annotated by human experts. These experts were simulating demanding and specific decisions in their reaction patterns during their annotation. The results show that the model only dropped 8% on the F1-score metric, which is very acceptable, given that the reaction patterns of the users were not previously observed on the training set, but they were original to this human evaluation.

Given that alerts can be organized hierarchically, the alerting tool can provide the capability to users to view all similar alerts within the same hierarchy with a simple click when an alert is presented. An example of such hierarchies includes the grouping of alerts, where issues related to CPU are designated under the "CPU" hierarchy, and those concerning networks are grouped in the "Network" hierarchy. Furthermore we tried to leverage the potential of these hierarchies for constructing a machine learning model that utilizes hierarchical representations. We delved into the historical datasets however, our examination revealed that user interactions were too sparse to support the development of a hierarchical machine learning model. On the contrary, employing a feature vector representation proved to be a viable alternative, particularly when augmented by the overflow mechanism.

6. Conclusions & future work

Alert fatigue is an increasingly pivotal topic in cloud monitoring tools which cannot be easily addressed by setting thresholds in the hundreds of monitored metrics from both beginners and experienced users. Alert filtering can be reduced in a binary classification problem that takes decision if an alert should be presented or not to the user based on some features that describe the alerts. Furthermore, in our research we have seen that groups of users with similar levels of experience in the monitoring tools have similar patterns of behavior and selection of the alerts. We leveraged this observation and we built an ensemble of binary classifiers. These classifiers correspond to the different levels of user experience. In this way our proposed methodology filters and presents to the users only the alerts that they would have selected if they had carefully inspected all of them. For the binary classifiers a random forest approach was adopted, by demonstrating their superior accuracy compared to other machine learning methods and without incurring prolonged training and inference times seen in more complex approaches like deep learning techniques.

Future endeavors aim to integrate the accurate alerting system in a next-generation network architecture, which will have as a goal to further automate the monitoring system. In particular, we aim to incorporate our proposed smart monitoring tool, as a means to achieve the necessary network assurance following the intent based networking paradigm to further reduce the intervention of network administrations, and thus to alleviate the alert fatigue problem. Thus, we aim to enhance our proposed methodology, by including fault compensation techniques, in order to reduce any hectic manual labor associated with monitoring, configuration and management of next-generation networks.

CRediT authorship contribution statement

Fotios Voutsas: Writing – original draft, Visualization, Validation, Software, Methodology, Data curation. **John Violos:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **Aris Leivadeas:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgments

We thank Andrew Maguire for the helpful discussions, Netdata for providing log files from Netdata Cloud, and the Netdata administrators that spared their time on annotating and validating our results. We would also like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC), grant No. RGPIN-2019-05250, for supporting in part our work.

References

[1] B. Varghese, R. Buyya, Next generation cloud computing: New trends and research directions, Future Gener. Comput. Syst. 79 (2018) 849–861.
[2] M.R. Mesbahi, A.M. Rahmani, M. Hosseinzadeh, Reliability and high availability in cloud computing environments: a reference roadmap, Human-centric Comput. Inf. Sci. 8 (1) (2018) 20.
[3] A.M. Fahad, A.A. Ahmed, M.N.M. Kahar, The importance of monitoring cloud computing: An intensive review, in: TENCON 2017 - 2017 IEEE Region 10 Conference, 2017, pp. 2858–2863, ISSN: 2159-3450.
[4] X. Zheng, A. Leivadeas, Network assurance in intent-based networking data centers with machine learning techniques, in: 2021 17th International Conference on Network and Service Management, CNSM, 2021, pp. 14–20.
[5] A. Leivadeas, M. Falkner, Autonomous network assurance in intent based networking: Vision and challenges, in: 2023 32nd International Conference on Computer Communications and Networks, ICCCN, 2023, pp. 1–10.
[6] A. Leivadeas, M. Falkner, A survey on intent-based networking, IEEE Commun. Surv. Tutor. 25 (1) (2023) 625–655.
[7] L. Turgeman, Y. Avrashi, G. Vagner, N. Azaizah, S. Katkar, Context-aware incremental clustering of alerts in monitoring systems, Expert Syst. Appl. 210 (2022) 118489.
[8] M.E. Aminanto, T. Ban, R. Isawa, T. Takahashi, D. Inoue, Threat alert prioritization using isolation forest and stacked auto encoder with day-forward-chaining analysis, IEEE Access 8 (2020) 217977–217986.
[9] Netdata: Monitoring and troubleshooting transformed, URL https://www.netdata.cloud/.
[10] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, M. Wolf, Monalytics: online monitoring and analytics for managing large scale data centers, in: Proceedings of the 7th International Conference on Autonomic Computing, ACM, 2010, pp. 141–150.
[11] W. Pourmajidi, J. Steinbacher, T. Erwin, A. Miranskyy, On challenges of cloud monitoring, 2018, http://dx.doi.org/10.48550/arXiv.1806.05914, URL arXiv:1806.05914[cs].
[12] G. Aceto, A. Botta, W. de Donato, A. Pescapè, Cloud monitoring: A survey, Comput. Netw. 57 (9) (2013) 2093–2115.
[13] J.S. Ward, A. Barker, Observing the clouds: a survey and taxonomy of cloud monitoring, J. Cloud Comput. 3 (1) (2014) 24.
[14] X. Wang, X. Yang, X. Liang, X. Zhang, W. Zhang, X. Gong, Combating alert fatigue with AlertPro: Context-aware alert prioritization using reinforcement learning for multi-step attack detection, Comput. Secur. 137 (2024) 103583.
[15] J.G. Baseman, D. Revere, I. Painter, M. Toyoji, H. Thiede, J. Duchin, Public health communications and alert fatigue, BMC Health Services Res. 13 (1) (2013) 295.
[16] S. Sen, W. Geyer, M. Muller, M. Moore, B. Brownholtz, E. Wilcox, D.R. Millen, FeedMe: a collaborative alert filtering system, in: Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work, CSCW '06, ACM, 2006, pp. 89–98.
[17] Y.-H. Su, M.C.Y. Cho, H.-C. Huang, False alert buster: an adaptive approach for NIDS false alert filtering, in: Proceedings of the 2nd International Conference on Computing and Big Data, ICCBD 2019, ACM, 2019, pp. 58–62.
[18] A. Pecchia, D. Cotroneo, R. Ganesan, S. Sarkar, Filtering security alerts for the analysis of a production SaaS cloud, in: 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, 2014, pp. 233–241.
[19] F. Voutsas, J. Violos, A. Leivadeas, Filtering alerts on cloud monitoring systems, in: 2023 IEEE International Conference on Joint Cloud Computing, JCC, 2023, pp. 34–37.

[20] T. Grossman, G. Fitzmaurice, R. Attar, A survey of software learnability: metrics, methodologies and guidelines, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '09, ACM, 2009, pp. 649–658.
[21] T.N. Poly, M.M. Islam, M.S. Muhtar, H.-C. Yang, P.A.A. Nguyen, Y.-C.J. Li, Machine learning approach to reduce alert fatigue using a disease medication–related clinical decision support system: Model development and validation, JMIR Med. Inf. 8 (11) (2020) e19489.
[22] N. Khreis, A.S.M. Lau, A. Al-jedai, S.M. Al-Khani, E.H. Alruwaili, An evaluation of clinical decision support and use of machine learning to reduce alert fatigue, Int. J. Comput. Commun. Eng. 8 (1) (2019) 32–39.
[23] A. Zien, N. Krämer, S. Sonnenburg, G. Rätsch, The feature importance ranking measure, in: W. Buntine, M. Grobelnik, D. Mladenić, J. Shawe-Taylor (Eds.), Machine Learning and Knowledge Discovery in Databases, in: Lecture Notes in Computer Science, Springer, 2009, pp. 694–709.
[24] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[25] X. Li, Y. Wang, S. Basu, K. Kumbier, B. Yu, A debiased MDI feature importance measure for random forests, in: Advances in Neural Information Processing Systems, vol. 32, 2019.
[26] B.P. Salmon, W. Kleynhans, C.P. Schwegmann, J.C. Olivier, Proper comparison among methods using a confusion matrix, in: 2015 IEEE International Geoscience and Remote Sensing Symposium, IGARSS, 2015, pp. 3057–3060.

Fotios Voutsas works as a Software Engineer at Netdata Inc. and is currently a Graduate student at Harokopio University of Athens, Department of Informatics and Telematics in the Master Program (M.Sc.) MPhil in Computer Science and Informatics. He received the B.Sc (hons.) degree from Harokopio University of Athens, Department of Informatics and Telematics, Greece, in 2023. His research interests include Network Engineering, Cloud/Edge Computing and Machine Learning.

John Violos works as a senior researcher at the Center for Research & Technology Hellas. He is also adjunct professor at Harokopio University of Athens and at National and Kapodistrian University of Athens. His previous positions include: Researcher in the Dept. of Software Engineering and Information Technology at École de Technologie Supérieure, Université du Québec in Montreal, Canada; Researcher at National Technical University of Athens, Greece. He was a member of the European Commission's Digital Single Market working group on the code of conduct for switching and porting data between cloud service providers. He has participated as work package leader, task leader and researcher in more than eleven research projects funded by the European Union, Korea and Canada. He has more than fifty publications in top-tier conferences and journals. He received the Best Paper Award of the IEEE CISOSE 2023 Conference, the Best Paper Award of the IEEE iThings 2021 Conference, the Best Course Teaching by an adjunct professor Award of Harokopio University of Athens 2021 and Thomaidion Award 2017.

Aris Leivadeas is currently an Associate Professor with the Dept. of Software and Information Technology Engineering at ETS. From 2015 to 2018 he was a postdoc in the Dept. of Systems and Computer Engineering, at Carleton University, Ottawa, Canada. In parallel, he worked as an intern at Ericsson and then at Cisco in Ottawa. He received his diploma in Electrical and Computer Engineering from University of Patras, Greece in 2008, the M.Sc. degree in Engineering from King's College London, UK in 2009, and the Ph.D. degree in Electrical and Computer Engineering from National and Technical University of Athens, Greece in 2015. His research interests include Cloud Computing, IoT, and network optimization and management. He received the best paper award in ACM ICPE '18 and the best presentation award in IEEE HPSR '20.