2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE) | 979-8-3503-5388-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISSRE62328.2024.00014

Exploring Hierarchical Patterns for Alert Aggregation in Supercomputers

Yuan Yuan*[1], Tongqing Zhou*[1], Xiuhong Tan†[2], Yongqian Sun[3], Yuqi Li[1, 4], Zhixing Li†[1], Zhiping Cai[1] and Tiejun Li[1]

[1]National University of Defense Technology, {yuanyuan, zhoutongqing, lizhixing15, zpcai, tjli}@nudt.edu.cn
[2]Changsha University of Science & Technology, tanxh@stu.csust.edu.cn
[3]Nankai University, sunyongqian@nankai.edu.cn
[4]National Supercomputer Center in Tianjin, liyq@nscc-tj.cn

***Abstract*—Alongside the high performance built on massive hardware, ever-larger computer systems bear tons of hardware alerts every day during reliability maintenance. Based on an exploratory study on a representative supercomputer system, this work first characterizes supercomputer alerts as an overload of continuous bursts for the operators. Yet, existing similarity-based aggregation solutions, tuned for in-band textual alerts, are myopic by finding dissimilar representatives instead of looking into the semantics in the supercomputer context. To fill the void of supercomputer alert aggregation, we propose the SuperAgg framework to extract the hierarchical patterns of real-world alerts and use them for online alert management. SuperAgg jointly integrates unsupervised state detection of time series and expert analysis to successfully discover 4 categories of sensor-tier alert patterns and exploits primary-and-secondary statistics between sensors for system-tier correlation patterns. With such extracted knowledge, SuperAgg then identifies the formulated patterns online and uses spatiotemporal combined strategies to reduce the alert influx. Evaluations on alerts generated from a production supercomputer show that SuperAgg provides over 98% aggregation rate and significantly higher aggregation accuracy (over 83.8% and 43.2% on different datasets) than 3 baselines. Production deployment further demonstrates its effectiveness from the perspective of system operators. The source code is available at: https://github.com/Txh-User/SuperAgg.**

***Index Terms*—Supercomputer reliability, alert management, time series analysis, AIOps.**

I. Introduction

Nowadays, high-performance computer systems are characterized by ever-growing computation, network, and storage resources [1] [2], especially true for large-scale data centers and supercomputer systems in the (pre)-exascale computing era. For example, the Tianhe-2A supercomputer system integrates 16K computing nodes, each with 64GB RAM and 400Gbps external one-way bandwidth [3]. Detecting abnormalities and failures of such massive hardware in time is critical, among other things, for the reliability of the system and providing continuous services. For this, out-of-band monitoring and diagnostic infrastructure are commonly deployed in production systems, collecting hundreds of indicators (e.g., voltage, temperature, and humidity) for individual components within each node [3] [4]. Once the sensory data violates pre-defined thresholds, alerts would be issued to notify the operators, facilitating the primary avenue for accommodating the behavior of supercomputers.

In this paper, we use a production supercomputer as the representative to collect facts and study management on massive alert data, considering that supercomputer is undoubtedly the most computation-intensive HPC infrastructure. Thousands of boards in a supercomputer carry tons of sensors, generating a huge amount of alerts. We find the raw alerts in a production supercomputer, even after a black list suppression, could still easily reach a prohibitive number of around 200 per 10 minutes. Unlike occasional alert storms in existing online service system [5], we observe that the overwhelming alerts in supercomputer systems manifest a stream of bursts, which we call **alert overload** (§ III-C). As identified in our field study, an experienced operator of the supercomputer system may spend as many as 6 hours per day mitigating all the alerts s/he believes to be important, which relies highly on subjective analysis and is prohibitive for everyday operations. This calls for automatic alert aggregation before reporting to remove redundant alerts and elaborate on the key ones, even at the cost of losing some inherent important signs, because it is favorable to retain the concise and useful ones instead of the cumbersome, thus useless, ones.

The existing efforts for alert aggregation/summarization all adopt a similarity-based strategy for unstructured in-band alerts [5]–[10]. Their basic idea is to pick out alerts that are dissimilar to each other. For this, one way is to use Jaccard distance on content [6], topology distance on locations [5], and temporal distance on time [10] of alerts for clustering and take the centroids as summarization. Some others release the alerts with different texts and occurrence frequency in sliding windows [7]. Nevertheless, similarity-based alert aggregation, focusing merely on comparing the content, is myopic in handling highly structural out-of-band alerts in supercomputers. Fig. 1 is an example taken from a real alert time series (§ IV-B), with the alert level frequently changing on the influx of new alerts. Especially, since alert levels are limited, for these alerts from the same board (same location), the changes are temporally quite small, making clustering and window-grouping in this window degenerate to random selection.

* Both authors contribute equally to this research.
† Corresponding authors.

2332-6549/24/$31.00 ©2024 IEEE
DOI 10.1109/ISSRE62328.2024.00014

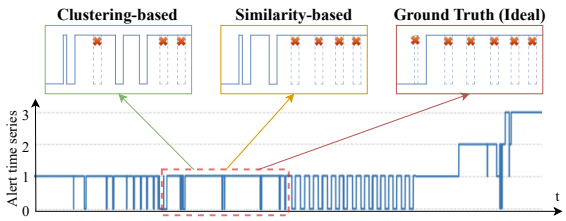


Fig. 1. The processing results of different alert aggregation methods on an example alert time series. The values in the y-axis indicate different alert levels, where $y > 0$ represents alert activation and $y = 0$ for no alert. The dotted lines in the upper boxes indicate the alert time subsequences (up/down triggered by an alert) removed by corresponding aggregation methods, and the red cross represents their deletion.

As illustrated with the ideal subsequence (red box in Fig. 1), merely the first alert is informative and shall be retained. Such refinement could only be attained with dedicated semantics of supercomputer alerts, telling which alerts are the outcome of redundancy or byproduct. Yet, unlike the in-band textual alerts, out-of-band alerts are issued without semantic information, making simple filtering rules unsuitable. Notably, observations gained from data in the Next Generation Tianhe Supercomputer (NG-Tianhe) indicate that 1) *the alert channel of a single sensor manifests continuous burst*, whereas 2) *channels of different sensors, both on the same and different boards, may share similar subsequences*. These motivate us to explore aggregation opportunities for supercomputer alerts from the sensor tier and system (cross-sensors) tier hierarchically.

We present the design of SuperAgg, the first framework for alert pattern learning (offline) and aggregation (online) in the supercomputer system. Essentially, SuperAgg learns hierarchical patterns by parsing sensor-tier alert semantics and spatial correlations from historical alert data in the production system. First, to overcome the pitfall of labeling tons of alerts, SuperAgg adapts an unsupervised state detection technique to find countable representative subsequences and jointly feed them to an expert group for offline analysis. Such a human-in-the-loop design for knowledge discovery is common in AIOps to facilitate interpretability, which is important for taking automatic operations in production systems. With this, we successfully summarize 7 typical sensor-tier patterns, formally categorized as stable, fake, wandering, and jittering, where the latter three are redundant and should be removed. Second, system-tier correlation between alerts issued from different sensors is modeled by estimating their temporal-lagging co-occurrences (e.g., a temperature alert on board 1 always happens after a voltage alert on its neighbor board), which extracts primary-and-secondary rules from historical alerts.

During online aggregation, two strategies of waiting and observing are invented by identifying the above redundant sensor-tier patterns and automatically suppressing alerts that are believed to be redundant. These refined alerts are further aggregated by merely releasing the precedent/primary alerts according to the spatial rules in the system tier. We highlight that our aggregation design, *totally and explicitly guided by expert knowledge and primary-secondary inference, would not incur additional false positives (i.e., discarding expert interested alerts) compared with a predefined black list*. Finally, we have constructed two datasets from the production supercomputer system and evaluated SuperAgg by comparing it with both a practical baseline and two similarity-based baselines. The major contributions are as follows.

- This is the first work to identify the alert overload pitfall of supercomputer alerts. An exploratory study on a production system brings out operator expectations and insightful observations for alert management.
- We present the novel design of SuperAgg for the offline discovery of hierarchical patterns in supercomputer alerts and their interpretable aggregation with dedicated strategies in real-time alert series.
- Evaluation on alert data of a production system shows SuperAgg's superiority in aggregation rate and accuracy, compared with 3 baselines. Real-world deployment qualitatively demonstrates it is favored by operators.

The rest of the paper is organized as follows: § II presents the related work. § III describes the exploratory study and observations on supercomputer alerts. § IV describes the overview and details of SuperAgg. In § V, we demonstrate the effectiveness and efficiency of SuperAgg by experiments based on real-world datasets. § VI discusses the SuperAgg. § VII concludes the paper.

II. Related Work

A. Alert Aggregation

With the development of Artificial Intelligence for IT Operations (AIOps), alert management has become an important part of AIOps, including alert aggregation [5]–[10], alert correlation [11] [12], and alert ranking [13] [14]. Wherein, the latter two could be used as a component for aggregation (e.g., Maximal Information Criteria correlation is used for aggregation in [8]). Here, we describe the representative alert aggregation methods, which are all similarity-based in design.

Lin et al. [6] use Jaccard distance to measure the similarity between textual alerts and perform graph-based clustering to find the principal incidents. Zhao et al. [5] introduce topological similarity into the Jaccard measure and leverage DBSCAN to attain a series of centroids as the summary of an alert storm. These works are criticized for ignoring the occurrence characteristics of alerts. Accordingly, Chen et al. [7] integrate the CBoW model and occurrence embedding for alert feature representation and similarity estimation, and reduce textual and occurrence similar alerts in the same window. As to the intrusion detection domain, aggregation is clustered based on alert groups (formed by interarrival time in [9] and temporal segmentation in [10]), instead of independent alerts, hoping to retain the local consecutive alert order that may form intrusion.

We note that existing efforts focus on aggregating unstructured alerts (system in-band alerts) by measuring their content similarities, which cannot effectively distinguish the structured hardware runtime alerts in supercomputer systems

that have similar attributes. More importantly, similarity-based aggregation ignores the semantic pattern of alerts, thus leading to the random suppression of important alerts.

B. Pattern Analysis for Time Series

The pattern of time series could be embodied as motif [15] and segmentation [16] [17] [18] in different contexts. Motif are repeated subsequences in multivariate time series (e.g., entomology). The recent VACOMI [15] method attains variable-length motif discovery by performing min-max-min cross-series distance calculation and comparison until consensus on a special subsequence. Segmentation, on the other hand, finds the change points in time series (e.g., IoT) for deducing the changes in the underlying process. For example, Kontrast [16] discerns erroneous software alterations by juxtaposing KPI time series data both antecedent to and subsequent to the implementation of software modifications. ClaSP [17] iteratively finds the best split for a subsequence by tuning a classifier to identify whether each pair of two adjacent segments has a similar shape. The SoTA method, Time2State [18], uses contrastive learning to efficiently segment time series with clear boundaries on the deep features.

These analysis processes are inspiring for understanding alerts in supercomputers. However, they are designed for handling continuous KPI/values, while the hardware alerts in this work are discrete and have already been full of change points (a new alert means the entrance of a new alert level). Dedicated adaptation for these techniques and domain knowledge are thus gathered to facilitate the analysis of supercomputer alerts.

III. Exploratory Study on Supercomputer Alerts

This section describes the practical facts, as well as observations on the tremendous amounts of hardware alerts in production supercomputers.

A. Alert Reporting

To monitor the runtime situation of the massive hardware resources in supercomputers, a hierarchical alert reporting mechanism is often adopted. Taking the production environment of *Tianhe* HPC systems as an example, ① the Board Management Unit (BMU) on each board is responsible for collecting alerts from the diverse sensors mounted on the board. ② In each chassis (a collection of boards), BMUs use SNMP-trap to actively report their detected alerts to the Chassis Management Unit (CMU), which parses and records the alerts in specific structure (§ III-B). ③ Each chassis's CMU periodically (e.g., 15s) reports the local alerts to the System Management Unit (SMU) for retention and rendering the comprehensive alert situations of the supercomputer system to the operator front end.

Typically, in the *Tianhe* HPC systems [3], each board is designed to have over 200 sensors for key components like CPU, memory, and I/O, covering temperature, voltage, current, and moisture, among many other aspects. Black or block list, as a rule-based mechanism, is often used to surpass the reporting of some disinterest sensors according to the operators' experience. Unfortunately, as we will see in § III-C, even after filtering out a large portion of channels with the black list, the system is still exposed to thousands of alerts every day owing to the massive monitoring basis in the supercomputer.

B. Alert Data

As aforementioned, alerts are triggered in BMUs and formalized in the corresponding CMUs. An example of the principle attributes of the structured alert data is shown in Table I. Each alert could be located by "LF" (row-chassis) and board name and is featured with a "type" indicating its severity and a "value" for the specific sensory data.

TABLE I
AN EXAMPLE FOR HARDWARE ALERTS IN A PRODUCTION SUPERCOMPUTER SYSTEM.

Sensor ID	**Board Name**	**IP**	**LF**
Temp_3	S*09	192.*.*.*	R3-P*
Volt_1	C*01	192.*.*.*	R3-P*
SensorVal	**Type**	**RecStat**	**Timestamp**
68	NC	0	2023-04-01 00:22:42
11.8	DNC	1	2023-04-01 00:23:02

In common practices, out-of-band (hardware) alerts are categorized into 3 levels[1], i.e., NC (Non-Critical), CR (CRitical), and NR (Non-Recoverable), with three underlying trigger thresholds for each sensor [19]. The switching of each alert level is controlled by two types of alerts, one for issuing it and another for dismissing it. For example, the BMU may be set to trigger an NC-level alert (type *NC*) when the temperature is > 60 and dismiss it (type *DNC*) when the temperature returns to 58. When there are alerts not being dismissed, the operators at the front end would assess the severity and urgency based on the alert level and take necessary measures to maintain the big machine in a good state.

C. Alert Overload

We collect the reported alert data of 4K boards in the *NG-Tianhe* system from 2023/01/28 to 2023/06/06 (130 days in total) via its unified monitoring system. To study and understand the underlying alerts, brute-force suppression rules (e.g., filtering out NC-level alerts or merely monitoring a few empirically useful sensors), which may overlook important hardware issues, are not activated. This provides us with over 3.66 million raw alerts.

1) Not just one burst: To understand the scale of alerts, we calculate the per-day and per-10m alerts statistics of the system. Fig. 2 shows the cumulative distribution function (CDF) of the alerts received by the system, which, if not being processed, would be all reported to the front-end operator. For almost every tested day, the system faces at least 10K alerts and the scale goes beyond 25K for 40% days of the runtime collection, as shown in Fig. 2(a). When referring to the finer

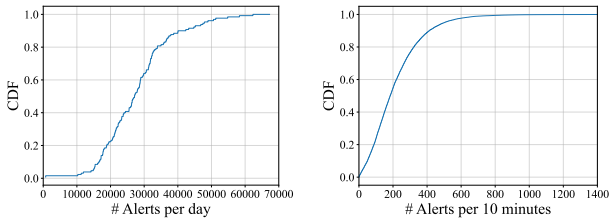
[1]There are 3 positive and 3 negative levels in total. W.l.o.g., we only discuss the positive ones hereafter for simplicity.

granularity in Fig. 2(b), we can see that the system has to deal with over 500 and 200 alerts in 90% and 50% of the observed 10-minute time slot, respectively.

As we will revisit in § III-C2, most experienced operators could only afford to handle a maximum of 10 alerts in 10 minutes, so 5~20 focused and full-time operators are needed to support real-time alert processing.

> Observation: The production supercomputer system faces continuous disturbance of alerts, named alert overload. Different from the occasional alert burst reported in data center systems [5], alert overload is a stream of bursts overwhelming the operators.

What's worse, the historical alerts not handled in time would accumulate to make the burden more excessive.



(a) CDF of alerts per day (b) CDF of alerts per 10m

Fig. 2. Illustration of the continuous stream of alerts.

2) A field-survey on handling alerts: We further design a questionnaire[2] to investigate operators' daily experiences in handling our supercomputer's overloaded alerts. Eighteen operators from different service teams are invited to participate in the survey, each with operation experiences of at least 2 supercomputers.

By analyzing the collected questionnaires, we find that most engineers (61.11%) adopt the rule-based strategy, focusing on high-level alerts only, to suppress the alerts. According to the feedback, they are aware that such strategies may cause overlooking signs of failure and lag in operations and maintenance. Although 66.67% of them reflect being able to deal with 10 alerts in less than 10 minutes, such compromise (i.e., overlooking and lagging) is still reported to be unavoidable, due to the significant number of alerts (66.67%), difficulties in analyzing multi-dimensional alert information (33.33%), and the complex correlation between alerts (38.89%). Unfortunately, this would still cost them an average of 2.11 hours per day, with most of the alerts turning out to be not important.

D. Observations on Aggregation Opportunities

Fig. 3 shows the alert time series of three different sensors during a day (series construction will be explained in § IV-B), with the first two from the same board and the 3rd one from another board.

[2]https://www.wjx.cn/vm/YIIOA5Y.aspx

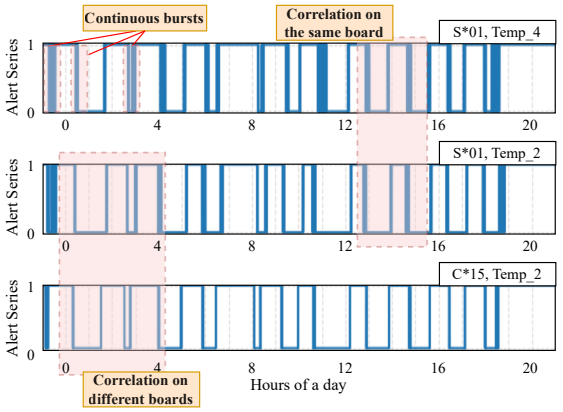


Fig. 3. Illustrative examples of the continuous bursts of sensor-tier alerts and spatial correlation among system-tier alerts. The continuous bursts in the "red boxes" are non-exclusive situations in our system.

> Observation: Underlying the overload of the system, single sensors issue continuous alert bursts (intensive alert level switching in the consecutive dotted box).

Given this, sensor-tier alert aggregation is critical to mitigate the overload from the sensory root.

> Observation: The alert series of sensors from both the same board and different boards often show similar trends at some periods (the dotted boxes across different series).

The following occurrence of alerts on different sensors indicates spatial correlation among them, which could be utilized for elaborating the root alert and suppressing the others.

IV. Design of SuperAgg

Given the observations above, we are inspired to explore the opportunities of aggregating alerts from two tiers, one for *local alerts within a sensor* and one for *global alerts among sensors of different boards and chassis*.

A. Framework Overview

An overview of the SuperAgg framework is shown in Fig. 4. SuperAgg works in two stages with the offline stage learning knowledge from historical alerts of our production supercomputer and the online stage performing aggregation based on the knowledge. The learning process involves 1) a *sensor-tier alert pattern modeling* component for discovering alert patterns for all the sensors independently based on self-supervised pattern detection and human-in-the-loop analysis (See § IV-C); and 2) a *system-tier correlation pattern modeling* component for summarizing primary-and-secondary rules of alerts in different sensors (See § IV-D). The learned alert patterns and correlation rules are then fed to strategy-based and spatial-based aggregation engines for inner-channel and inter-channel alert reduction in the online stage.

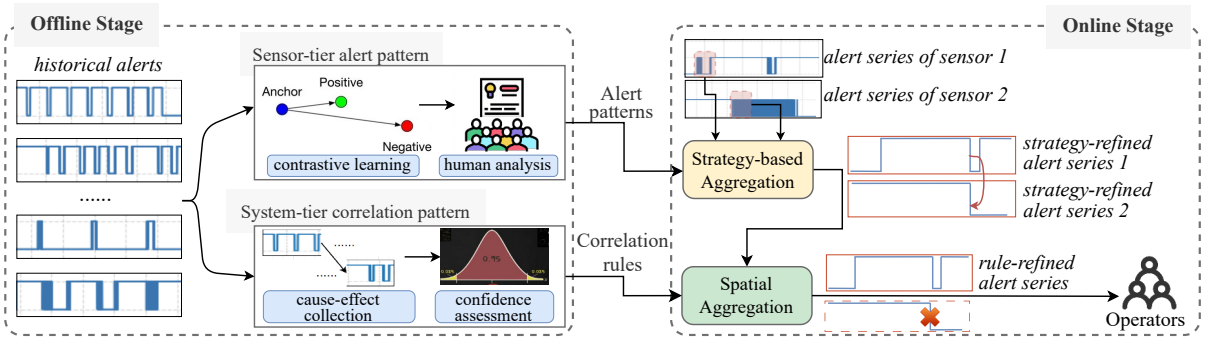


Fig. 4. The framework of SuperAgg, wherein two examples of alert time series are provided to illustrate online aggregation.

Note that offline pattern modeling, both sensor-tier and system tier, doesn't involve real-time operations, thus having no impact, on the production system. In practice, the modeling and learning could even be conducted on servers not in the production system as long as the historical data is recorded and stored on them, providing more opportunity for operators to discuss and elaborate on explainable aggregation guidelines.

B. Alert Pre-processing

During the training stage, historical alerts are transformed into time series with mechanism redundancy removed.

1) Transforming into time series: The raw alert data consists of discrete entries, as described in Table I. For pattern mining from the system perspective, we propose to transform the entries into a time series that records the system's evolving situation. Outlier alert entries with incorrect dates or exceptional readings are first excluded. We then use AL_i^t=0, 1, 2, 3 to denote no alert and level NC, CR, NR alert for sensor s_i (i is a system-wise index) at time point t. Given all the alert data of s_i, we use the "Type" attribute to switch the value of AL_i^t and gain the alert time series S_i=$\{AL_i^t\}_{t=0}^T$. In this way, a changing point in S_i indicates an event of alert issuing or dismissing. We will use $S_i^{p:q}$=1/2/3 to denote a subsequence of a time series where the sensor stays in the level of NC/CR/NR alert from time p to q.

2) Chain alerts reduction: During practice, we note that, since sensory values change continuously, the triggering of a high-level alert will unavoidably cause the issuing of low-level alerts. For example, if the thresholds of NC, CR, and NR of a temperature sensor are set to be 55, 60, and 65, then type *NC*, *CR*, *NR* alerts would be issued sequentially when the ambient temperature increases from 54 to 66. We call this phenomenon an **alert chain**. To mitigate the redundancy in the chain and ensure that the specific state (severity level) of each alert is accurately known to operators, we use a simple temporal filter that releases an alert only if no higher-level alerts appear in the time slot Δt (set to be 1s empirically) starting from its happening time:

$$\begin{aligned}\widehat{AL_i^t} =& Sign_{>0}(S_i^{t:t+\Delta t} - AL_i^t) \cdot AL_i^{(t-1)} \\ &+ Sign_{\leq 0}(S_i^{t:t+\Delta t} - AL_i^t) \cdot AL_i^t,\end{aligned} \tag{1}$$

where $Sign_{>0} = 1$ if there is an element in $S_i^{t:t+\Delta t} - AL_i^t$ is bigger than 0. This process is adopted for both offline learning and online aggregation.

C. Sensor-tier Alert Pattern Modeling

Observation from Fig. 3 reveals the intensive alerts in the series, which may inherently be event-triggered patterns. Figuring them out will help us understand the impact on the system and retain the important alerts. Let operators manually look into and analyze tons of alert series is not feasible, not to mention that new patterns may appear under new workloads and dynamic maintenance. As a remedy, we propose to automatically elaborate on a few candidate patterns for affordable human analysis.

1) Pattern Detection: As labeling the alerts is prohibitive for operators, pattern detection should be performed in an unsupervised way. We propose to adapt the state-of-the-art Time2State technique [18] for this purpose. As shown in Fig. 5, it uses contrastive learning to learn a state/pattern encoder $en(\cdot)$ that could detect patterns for alert time series. For the paper to be self-contained, we describe the brief workflow and adaptation of Time2State. On one hand, it stresses that neighbor alert subsequences falling in consecutive time windows are more likely to be with the same pattern. Formally, to encode these neighbor alert subsequences into close embeddings/vectors in the feature space, the following loss function is used:

$$\mathcal{L}_{pos} = -log(\delta(en(S^{t_i^m:t_i^m+w};\theta)^T \cdot en(S^{t_j^m:t_j^m+w};\theta))), \tag{2}$$

where θ is the network parameter for the encoder, δ is the sigmoid function, and t_i^m and t_j^m are the starting time of alert subsequence in two consecutive time windows. By performing several sampling (i.e., several (i, j) pairs)) in m places of the alert series, this loss optimizes the encoder on pulling together the representation of alert subsequences with the same pattern.

On the other hand, it considers that alerts falling in non-consecutive windows, sampled from different places in the alert series, are more likely to be with different patterns. To train the encoder for learning pattern differences, the following loss function is used:

$$\mathcal{L}_{neg} = -log(\delta(f_i^T \cdot f_k)), \tag{3}$$

where t_i^m and t_k^m are the starting time of two alert subsequences sampled from two random places in the alert series, and $f_i = \frac{1}{n_s} \cdot \sum_{i=1}^{n_s} en(S^{t_i^m:t_i^m+w}; \theta)$ denotes the average embedding vector for n_s sampling at the same two places. Similar to $\mathcal{L}_{pos}$, $\mathcal{L}_{neg}$ optimizes the encoder on pushing away the representation of alert subsequences with different patterns, i.e., telling the differences in alert patterns. One may argue that, in practice, alerts in randomly sampled time windows are not necessarily with different patterns (i.e., false negative in $\mathcal{L}_{neg}$). We emphasize strengthening the learning of alerts of the same pattern (i.e., with $\mathcal{L}_{neg}$)) will complement such false negatives and maintain the learning effectiveness.

The state encoder of Time2State for regular time series, with independent channels and short intervals, may miss the uncharted patterns in multi-channel and complex supercomputer alerts. Hence, we tune the encoder with context knowledge during adaption. Specifically, positive sampling should be within the same pattern for correct feature representation, so SuperAgg sets the sampling length upper bound as the system inspection period (e.g., 1 hour in the tested production system), during which period the behavior of the hardware is believed to be similar. Accordingly, for the parameter of sampling window size w, moving step $step$, and sampling number k, we set them according to w+$k \cdot step \leq 3600s$.

Based on these parameter settings, SuperAgg fine-tunes the encoder by sampling and training on each sensor's one-day alert data each time until learning every sensor's data on 130 days. This helps to avoid overfitting on a single channel of one sensor. It takes 270s (15 epochs) to learn the dedicated encoder for supercomputer alerts. We then infer potential patterns in all the time series with the encoder and regulation that each pattern shall appear in at least two different sensors. As shown in the example of Fig. 5, This yields the detection of 7 different patterns.

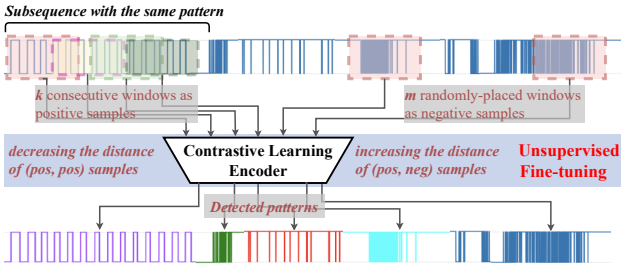


Fig. 5. Fine-tuning Time2state encoder for pattern detection.

2) Human-in-the-loop Modeling: The detection phase provides 7 different patterns, but their semantic meaning is unclear, which is unhelpful in deciding how to deal with them. Given that manually analyzing 7 patterns is lightweight, SuperAgg then carries out a human-in-the-loop pattern labeling phase to figure out the inherent event of each pattern with a focused discussion. *Note that integrating expertise knowledge is only for elaborating on interpretable guidelines in the offline pattern learning, which is a one-pass step, not used in the online automatic aggregation.* According to the feedback, we group them into four categories, including one stable, two fake (issuing and dismissing), two wandering (up and down), and two jittering (up and down) patterns.

Stable pattern. As shown in Fig. 6, the expert group of operators points out that a sufficient-long time (i.e., δ) period with at most one alert message (i.e., the number of fluctuation $N_{fl} \leq 1$) could be considered as stable, which is believed to be the normal and ideal behavior of sensors.

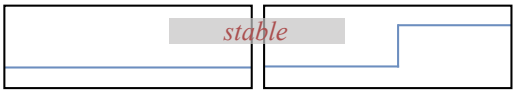


Fig. 6. Example and formulation of the stable pattern.

Formally, we could denote the stable pattern as follows, where len_j^p is the duration for a sensor staying in the j-th alert level for the p-th time:

$$N_{fl} \leq 1, len_j^p \gg \delta,\ j \in [0,1,2,3], p \in [1,...,N_{fl}] \tag{4}$$

Fake pattern. The experts categorize the two patterns of *consecutive spikes* shown in Fig. 7 as fake ones, mainly owing to the manufacture deficiencies of sensors. As a spike, the alert level switching in this pattern is instantaneous (i.e., $len_j^p < \delta$ with p the index of number of fluctuation). By using "fake", we mean that, after a period of fluctuation T_{fl}, it will always return to the original alert level, which has $AL_i^t = AL_i^{t+T_{fl}}$.

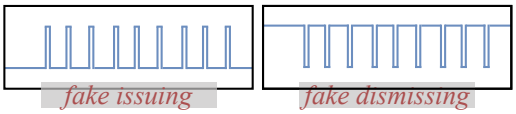


Fig. 7. Example and formulation of the fake pattern.

Formally, we could denote the fake pattern as follows:

$$AL_i^t = AL_i^{t+T_{fl}}, T_{fl} = \sum_{p=1}^{N_{fl}} len_j^p \tag{5}$$
$$len_j^p < \delta, j = [0,1,2,3].$$

Wandering pattern. This is identified as a similar pattern of *Fake*, i.e., also characterized as a series of spikes. The difference is that such a fluctuation is a wandering process before entering into a different alert level, so it has $AL_i^t \neq AL_i^{t+T_{fl}}$. The experts owe this phenomenon to the hardware sensory value falling in critical intervals, e.g., when the temperature ranges between 59 and 60 with an alert threshold of 60.

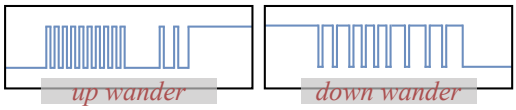


Fig. 8. Example and formulation of the wandering pattern.

Formally, we could denote the wandering pattern as follows:

$$AL_i^t \neq AL_i^{t+T_{fl}}, T_{fl} = \sum_{p=1}^{N_{fl}} len_j^p \\ len_j^p < \delta, j = [0, 1, 2, 3]. \quad (6)$$

Jittering pattern. It is identified as a wave-shape time subsequence, with significantly more fluctuations than pattern *Stable* each wave significantly longer than the spike in pattern *Fake* and *Wandering*. Like pattern *Wandering*, this pattern is considered to be caused by the slight sensory value changing around an alert level, but more often with the loading, execution, and scheduling of workloads.

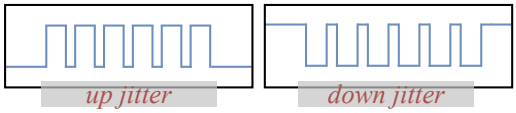


Fig. 9. Example and formulation of the jittering pattern.

Formally, we could denote the jittering pattern as follows:

$$N_{fl} \gg 2, len_j^p \gg \delta, \ j \in [0, 1, 2, 3], p \in [1, ..., N_{fl}]. \quad (7)$$

Note that, except for the stable pattern, the other patterns all involve semantic redundancy whose removal could yield concise alert data to the operators. We will explore these opportunities in § IV-E.

D. System-tier Correlation Pattern Modeling

Recall the observation on spatial correlation in § III-D, SuperAgg also tries to reduce the spatial redundancy by understanding correlations among different alert time series. For this, a *directional Apriori* method is used to mine the correlation rules by estimating the statistics of alert co-occurrences.

To lead to reasonable aggregation, we propose to find primary-and-secondary rules from the occurrence chain of alerts on different sensors. Intuitively, if type NC alert frequently appears on sensor s_i shortly after type CR alert on sensor s_j, then the latter is the principal manifestation of the inherent fault and could be considered the root cause of the former one. By only reporting and addressing the primary alert, the subsequent alert shall also be suppressed. Hence, SuperAgg first constructs a collection of primary-and-secondary alert groups. For each alert in s_i, SuperAgg checks whether its happening time is within the time window (empirically set to 10 minutes) of any existing alerts that have not been dismissed in the system, and if so, adding it to the effect group of the corresponding precedent alert. Then the support and confidence of correlations of a precedent alert and all its effects in the group are calculated as in Apriori [20].

It takes 45s to generate rules for our 130-days historical alerts. Each raw rule is represented as $Rule_i = [(c_1, b_1, AL_1) \rightarrow (c2, b2, AL_2), conf]$, wherein c_i and b_i are the chassis and board identifier of an alert, and $conf$ is the confidence in a rule. By retaining rules beyond a certain confidence value, a list of correlation is attained in the form of $\mathcal{R} = \{Rule_1, Rule_2, ..., Rule_L\}$, where L is the number of rules.

E. Pattern-aware Online Alert Aggregation

The two modeling components provide patterns and rules during offline learning, which are then used as the knowledge base for sensor-tier and system-tier online aggregation, respectively.

1) Sensor-tier aggregation: Different from the offline posterior pattern modeling, *SuperAgg has to identify and handle the online alert streaming in real time*. The crux in making the decision is the parameter δ, i.e., fluctuations $\leq \delta$ are either *Fake* or *Wandering*, while those $\gg \delta$ are *Jittering*. SuperAgg refers to the statistics in historical data to set the bounds, that is, denoting δ as the average of all the fake and wandering spikes' length and $\gg \delta$ as the average of all the jittering waves' length, which are around 10s and 30s in our context. SuperAgg then uses a strategy of *silent awaiting* in cases where alert levels switching $< 10s$ and strategy *see&suppression* in cases where alert levels retain unchanged $\geq 30s$ for aggregation.

Silent awaiting. Literally, this strategy keeps silent (i.e., temporarily not reporting the alert to operators) when receiving two consecutive alerts within δs until the fluctuation finishes. The rationale behind awaiting is that such a spike-alike fluctuation could be fake alerts or wandering before stable, so could only be identified when it stops fluctuating. We set the stop criterion as $len_j^{N_{fl}+1} \geq 3 * max(len_j^p)$ ($p \in [1, .., N_{fl}]$), i.e., the alert series stays on a level for at least 3 times the length of the largest spike. If the alert level at the finishing time point ($AL_i^{t+T_{fl}}$) is different from that of the starting time point, it will report $AL_i^{t+T_{fl}}$ to the operators (*wandering*), otherwise, no alert will be pushed to the operators (*Fake*).

See&suppression. This strategy records the wave fluctuation in the jittering pattern for a while (i.e., n=6 fluctuations) before deciding to suppress which level of alerts in the jittering. Specifically, the lengths of every wave at the two different alert levels are denoted as L_a=$[len_a^p, p \in [1, .., n]]$ and L_b=$[len_b^p, p \in [1, .., n]]$ ($a \neq b \in [0, 1, 2, 3]$). Then we use the independent-measures t-test to estimate whether there is a significant difference between L_a and L_b. If the difference in lengths is significant, then the alert level with the smaller average on length will be suppressed (e.g., reporting only A_I^a if $\mu(L_a) > \mu(L_b)$). Otherwise, both levels of alerts would be retained and reported.

2) System-tier aggregation: We explain the idea and rationale of spatial aggregation with a concrete case. Since boards in the same chassis share wind cooling subsystem, temperature sensors s_1 and s_2 on such two boards share the same ambient temperature controlling process. If s_1 is mounted closer to the wind inlet than s_2, then the former would issue an alert before

s_2 when the temperature goes beyond specific thresholds. Ideally, only the alert from s_1 is sufficient to remind the operators to take measures.

Given the rules $\mathcal{R}$ obtained in § IV-D, SuperAgg suppresses an alert if it has precedent primary alerts in window w (e.g., the alerts of s_1 and s_2 are primary and secondary in the above case).

$$Rule_k = [(*, *, AL_i) \rightarrow (*, *, AL_j), 0.8] \in \mathcal{R} \wedge (t_i - t_j) \leq w,$$

where t_i is the happen time of AL_i.

V. Evaluation

We evaluate the performance of SuperAgg by answering the following two questions based on data generated from the *NG-Tianhe* system:

(Q1) Performance of SuperAgg: How does SuperAgg perform in online aggregation?

(Q2) Effectiveness of the hierarchical design: Are sensor-tier and system-tier patterns and aggregations effective?

A. Settings

1) Datasets: As described in § III-C, we use the 130-day raw alert data in *NG-Tianhe*[3]. Since an overhaul is conducted in this period, we divide the data into 2 parts at the overhaul time point, which also helps to assess the generality. Note that it is difficult to evaluate the accuracy of aggregation (i.e., whether there are wrongly suppressed alerts) without labels on the ideal alert series while having operators annotate tons of alerts is infeasible. As a remedy, we propose to *put sentinels that shall not be removed during aggregation and use them for accuracy evaluation*. Specifically, the operators managed to pick out 1165 cases of sentinel alerts by manually referring to explicit context events and critical sensors in design. Table II shows the details of the datasets, wherein the minimum and the maximum number of per-day raw alerts are 10266 and 67262, respectively. In each dataset, we use the first three-quarters of alerts divided by time as the training set and the last quarter of alerts as the test set.

TABLE II
Details of Experimental Datasets.

Datasets	Time span	#Alerts	#Sentinel alerts
A	2023/01/28 $\sim$ 03/31	1552942	607
B	2023/04/01 $\sim$ 06/06	2115815	558

2) Metrics: We use *aggregation rate* to evaluate the alert reduction performance:

$$\frac{n_{before} - n_{after}}{n_{before}} \times 100\%,$$

which is the main impact factor of operators' experiences. Meanwhile, it is critical to make sure that the reduction is valid, instead of random removal, with key alerts for operation involvement and troubleshooting retained. The sentinel alerts have been extracted to provide an approximate estimation of the *aggregation accuracy* by the ratio of retained sentinels after aggregation (i.e., considering the sentinel as a sampling of the population of key alerts).

3) Baselines: Existing alert aggregation techniques could be categorized as rule-based methods [21] favored in practice, clustering-based methods [5] stressing global features, and similarity-based ones [7] counting on local correlation. Since there is no public implementation for the above methods in the context of supercomputer, we have implemented the above aggregation methods to construct three baselines by referring to the formulation and settings in their papers.

Rule-based [21]. It attains a naive aggregation by constraining the number of reported alerts on different levels for a concise alert list at the front end. According to the statistics in historical data, the upper bounds of allowed alerts for the NC, CR, and NR levels are set to be 400, 200, and 20. This also provides a similar aggregation rate to other methods, making it intuitive for comparison.

Clustering-based [5]. We adapt the clustering aggregation methods in [5] by extracting the physical locations of sensors and alert levels as discrete features and sensory values and timestamps as continuous features. A normalized distance is then calculated for the extracted features of each alert pair. DBSCAN clustering (ϵ=0.05 and $minpoints$=50) is performed to group the alerts in a window into clusters. Only the alerts corresponding to the centroids are finally released.

Window-based [7]. It suppresses a new alert with $AL_i^{t_1}$ if it is sufficiently similar to a precedent alert $AL_i^{t_0}$ and t_1-$t_0 < w$. We set the similarity threshold to 0.97 (normalized) and the window size to 10 minutes.

We implemented these approaches with Python and ran them on different CMU servers with 16GB memory. The settings are typical because such servers are widely used in commercial clusters and data centers.

B. Performance on Alert Aggregation

1) Aggregation rate: The aggregation rate is an important indicator of how much burden a method can relieve for the operators. Table III shows the comparison of the aggregation rate of SuperAgg and the baselines. We can observe that, even set with purposely high aggregation strength (i.e., significantly small bounds, and small similarity threshold to clustering centroid and precedent alert), all the baselines showcase inferior aggregation rates than SuperAgg. Operators only need to focus on the refined alerts.

The main reason is that the baselines ignore the semantic redundancy in the supercomputer's alerts. On one hand, the rule-based method's fixed bounds on allowed alerts would yield varied performance on days with different numbers of alerts and should be carefully set based on the experience of the operators and the realistic scenarios. Further, once the number of alerts at one level has met the threshold, later alerts, even important in operation, would unfortunately be

[3]Note that, as the first work on handling out-of-band alerts on large-scale computer systems, we cannot find any prior public datasets in this context. We believe that evaluation on the real data from a top-level supercomputer system is sufficiently representative.

ignored. On the other hand, we notice that the clustering method presents a significant aggregation rate decrease on dataset B, because the number of alerts each day is much different in B, making the clustering and inferring process unstable.

TABLE III
THE AGGREGATION RATE OF SUPERAGG AND BASELINES

Methods	Aggregation Rate (%)	
	Dataset A	Dataset B
Rule-based	97.10	97.74
Clustering-based	97.77	94.81
Window-based	97.87	97.33
SuperAgg (ours)	**99.04**	**98.64**

2) Accuracy: Then we check the aggregation accuracy by estimating the protection of sentinel alerts during redundancy removal or suppression. The experimental results on the accuracy of SuperAgg and the baselines are shown in Fig. 10. We note that, with similar performance on aggregation rate, our SuperAgg showcases a large margin on accuracy compared with all the baselines (i.e., at least 83.8% on dataset A and 43.2% on dataset B).

We owe such superiority to the pattern discovery and exploration of SuperAgg, which helps to automatically remove redundancy with expert-provided semantics. The clustering and window methods both rely on measuring the similarity of alerts, under the idea of de-duplicating similar alerts. This would cause many false aggregations as referring to the limited discrete alert levels may easily group two temporal neighbor alerts as duplicative. For instance, these two baselines would very likely suppress one of a group of NC alerts on different boards of the same chassis issued at the same window, though they provide load-balancing signs for specific boards. Besides, the rule-based method has a significantly lower accuracy than the other baselines, indicating that important sentinels are brutally removed when the daily number of alerts at each level reaches the set bounds.

SuperAgg has over 95% accuracy on both datasets, which indicates better generality than the baselines. The clustering-based and window-based methods suppress alerts proportional to the scale/density of the alerts, which would retain more data in the same time window on the dense dataset (B) than the sparse dataset (A), thus showing a higher probability of retaining sentinel alerts. Finally, an accuracy smaller than 100% means that mis-suppression could happen, which may cause later failure. While this is true, we note that the raw alerts are with an amount that is incapable for operators to handle, in which case more key alerts would be overlooked by exhausted operators. Hence, aggregation with slight mis-suppression is practically acceptable.

3) Time cost: This part tests the online alert processing time (delay) of different methods, which is critical for the timely troubleshooting of supercomputer operation. Before working online, SuperAgg requires offline unsupervised pattern learning, as explained in § IV-C1 and § IV-D to be 270s

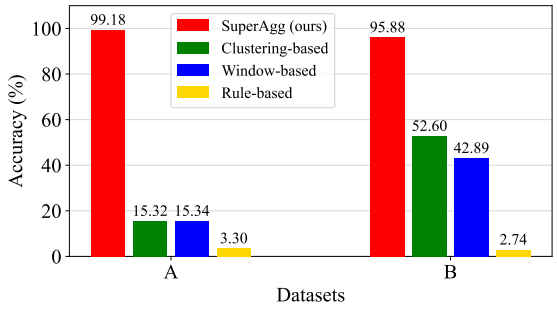


Fig. 10. The aggregation accuracy of SuperAgg and baselines.

for modeling patterns in sensor-tier and 45s for generating rules in system-tier on the 130-day trace, separately. Note that the offline stage causes no dynamic time cost to the online alert aggregation and the patterns could be updated in the background at any time with sufficient historical data.

The online processing time cost of SuperAgg is δ=10s because after receiving each alert, the engine will wait for δ to decide whether it is in the fake or wandering pattern. SuperAgg will neglect it if it is a *Fake* or *Wandering* while releasing it to or suppressing it from the operator according to the aggregation strategies. Considering that mainstream monitoring products, such as Azure Alerts [22], fire alerts in around 1 minute, the delay of 10s for aggregation is acceptable. As to the baselines, the time costs of window-based and rule-based baselines are $O(1)$, which is negligible by involving only a few distance estimation or number counting operations. In the tests, the average time cost of processing an alert for the clustering-based methods is 281s. This baseline requires performing clustering on alerts in a time window, thus causing additional delay. In the worst cases, the maximum delay for this method could be 345s.

Note that all the methods, except clustering, use a one-pass way in online alert aggregation, so the time costs would not scale with denser alerts. The clustering time would increase when more alerts happen in a time window.

C. Ablation Study

SuperAgg works on sensor-tier strategy aggregation and system-tier rule aggregation. This part conducts an ablation study on these two aspects of aggregation to test their effectiveness. Table IV shows the performance comparisons on two datasets achieved by SuperAgg, only sensor-tier aggregation (i.e., w/o system-tier aggregation) and only system-tier aggregation (i.e., w/o sensor-tier aggregation).

As shown, sensor-tier aggregation (strategy) provides a significantly higher aggregation rate and accuracy than system-tier aggregation (correlation rules). This owes to the more excessive redundancy of the single-channel alert time series than the cross-channel ones. For example, in a fake/wandering pattern (major patterns in the observed alerts), generally, $>$ 10 spikes appear, each with 2 alerts (one issuing and one dismissing), which generates $>$ 20 redundant alerts in

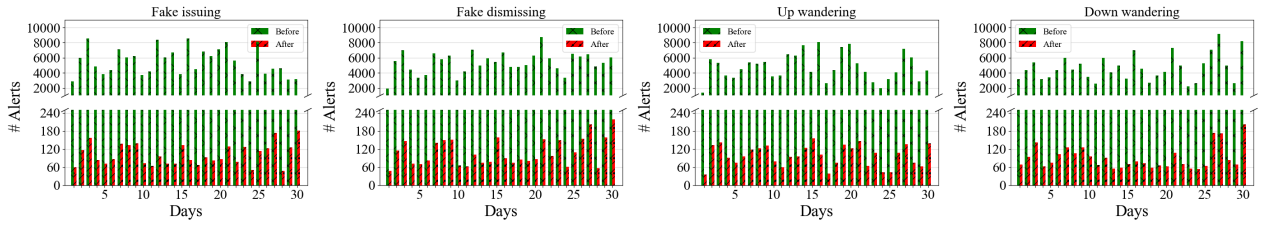


Fig. 11. Alert aggregation of the fake and wandering patterns based on the silent awaiting strategy.

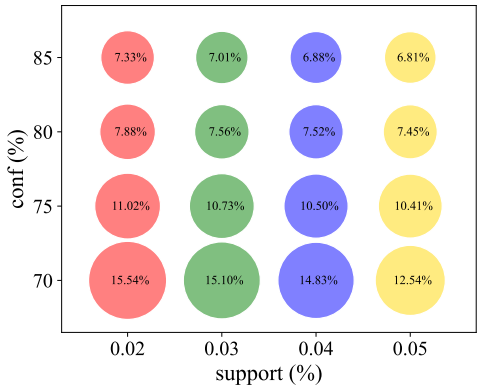


Fig. 12. The aggregation rate of system-tier aggregation under different confidence and support settings.

a 200s time window. Hence, over 98% of alerts (0.38 and 0.52 million alerts for datasets A and B, respectively) are suppressed by the sensor-tier aggregation. Notably, no sentinel is mis-removed during sensor-tier aggregation (accuracy 100%), because only the redundant alerts in each pattern are suppressed while the principal alerts (if there is a sentinel) are retained. Still, system-tier aggregation independently provides a modest accuracy performance compared to the results of the baselines.

It seems that the involvement of system-tier aggregation reduces 1%~4% of the accuracy by improving only around 0.2% aggregation rate. We emphasize that the relative improvements of rule aggregation (system) from strategy aggregation (sensor) in aggregation rate are 15% and 19% for datasets A and B, respectively, successfully reducing 60~100 alerts per day. Operators state in our qualitative study that such accuracy sacrifice for fewer alerts is favorable and they could always deactivate this aggregation dimension during the support of important tasks.

Furthermore, we also dig into the aggregation performance of strategy and rules independently. Wherein, we only examine dataset A, considering that the variation in the number of alerts per day in dataset B is unstable for straightforward observations.

Effectiveness of sensor-tier strategies. The fake and wandering patterns dominate the alert data collection, so we compare the number of alerts before and after the processing of the silent awaiting strategy as a representative. The results

TABLE IV
ABLATION STUDY RESULTS ON ONLINE AGGREGATION

Methods	A		B	
	AR (↑)	ACC (↑)	AR (↑)	ACC (↑)
w/o system-tier aggregation	98.88	**100**[*]	98.33	**100**[*]
w/o sensor-tier aggregation	8.85	20.00	21.79	41.33
SuperAgg	**99.04**	99.18	**98.64**	95.88

[*] Only redundant alerts in each pattern are removed, so if there is a sentinel in the pattern, it will be retained as the principal alert by the strategy.

of fake issuing, fake dismissing, up wandering, and down wandering patterns on dataset A are shown in Fig. 11. The number of alerts in each pattern is reduced significantly, with the max, min, and average aggregation rates of different patterns 99.03%, 94.24%, and 97.94%, respectively. Notably, even if the number of alerts for a given pattern exceeds 8000 a day, SuperAgg is still effective in reducing the number of alerts to less than 240.

Effectiveness of system-tier rules. Then we analyze the aggregation gains of different amounts of spatial correlation rules, tuned by parameters $support$ and $conf$. The result is shown in Fig. 12. We can see that larger $support$ and $conf$ generate fewer rules, and the number of rules is directly proportional to the aggregation rate in practice. The reduction in the number of alerts is at a minimum of 6.81% and a maximum of 15.54%. Given that the number of spatially correlated alerts is relatively low and operators take a cautious attitude on cross-sensor suppression, such an aggregation rate is acceptable.

D. Parameter study

1) Impact of δ: We further conduct a variable-control experiment on dataset A to explore the impact of parameter δ in the online alert pattern detection performance. Fig. 13 shows the overall performance by ranging δ in {2, 4, ... 20} seconds. We can see that, with the increasing of δ, SuperAgg's accuracy first decreases and then increases to relatively stable (8~12s). The aggregation rate increases with δ until receiving stable at 14~20s. The rationale is that very small δ may cause the strategy to be rigid on considering a subsequence to be *Fake* or *Wandering*, yielding a relatively low aggregation rate and high accuracy (more sentinels retained as fewer alerts aggregated). On the other hand, δ larger than the common length of *Fake* or *Wandering* (i.e., 12) would wrongly suppress the subsequences

TABLE V
IMPACT OF $support$ AND $conf$ ON THE NUMBER OF RULES

# Rules		*Support*			
		0.02	0.03	0.04	0.05
Confidence	70%	740	342	223	144
	75%	624	273	171	105
	80%	509	204	123	72
	85%	388	159	86	46

of *Jittering* using the silent awaiting strategy. During practices, we recommend carefully setting δ with the average value of observed spikes (e.g., 10s in our context).

2) Impact of $support$ and $conf$: As shown in in Table V, we also test the impact of $support$ and $conf$ on spatial rules by varying them in the range of {0.02%, 0.03%, 0.04%, 0.05%} and {70%, 75%, 80%, 85%}, respectively. When $conf$ is fixed, the number of rules decreases agilely with the increase of $support$, while it decreases more slowly with the increase of $conf$, indicating stronger sensitivity to the support value.

Fig. 14 shows the impact of these rule generation parameters on aggregation. With more rules, the relative aggregation rate increases first, decreases in the middle, and then increases with a relatively minor overall change, while the accuracy showcases a contrary trend. The time cost changes slightly with the variation of rules. The above phenomenon suggests that 1) setting $support$ as 0.04% and $conf$ as 70% can obtain a proper number of rules (225 ± 10), and maintain a proper performance on system-tier aggregation; 2) $support$ and $conf$ both impact the performance of system-tier aggregation.

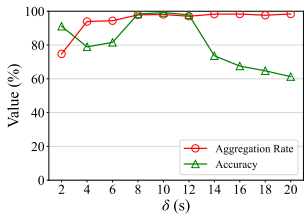


Fig. 13. Impact of δ

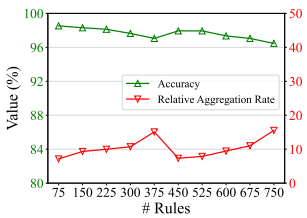


Fig. 14. Impact of the number of rules

VI. IMPLEMENTATION AND DISCUSSIONS

A. Qualitative Study

We have deployed SuperAgg in a production supercomputer system for a runtime test of 5 days. After a minute-level pattern learning and confirmation, it seamlessly begins online aggregation. According to the feedback from the field, operators #1 and #3 state that it is fantastic to hear fewer alert rings without worrying that important signs are missed. Operator #2 brings up the expectation to see the comparison between the raw alerts and the refined ones to understand how the removal ones are aggregated into the final alerts, which is an insightful idea for building an operation knowledge base.

One should note that the design of SuperAgg is *also suitable to alert management in data centers and clusters that rely on out-of-band monitoring for reliability*. Such scalability owes to its unsupervised and interpretable design in pattern learning. We ask for the community to adapt SuperAgg on their private hardware data for more research tests and discussions, instead of directly using our elaborated alert categories.

A concern on SuperAgg is that pivot signs or important alerts may be accidentally removed, which may cause negative impacts on the production system. We argue that, without aggregation, operators cannot find any if not all, in-time and important clues from cumbersome alert series, so SuperAgg significantly relives this situation. Meanwhile, the suppressed alerts are not deleted and the operators can find them in the database to investigate if a failure is correlated to them and recover such false-negative alerts in later aggregation.

B. Discussions

1) Limitations: One limitation of SuperAgg, as well as all the studies on alert management, is the lack of labeled data on the ground truth, making knowledge extraction and performance evaluation challenging. Although sentinels are set in our evaluation, it is just a subset of the estimation of the ideal alerts that manifest the fault or failure in hardware runtime. We note that a feasible remedy for such a pitfall is a scalable annotation tool for redundancy, state, and even root cause labeling of tremendous alert data. Besides, we could leverage the system log as context knowledge to prioritize alerts that happen during memory, disk, and OS faults.

2) Implication: The data generated in supercomputer systems is a representative form of big data. From the qualitative study of this work, we find that elaborating information that is affordable to users (operators in our cases) is more favorable than maintaining absolute integrity because users have to randomly drop any information when they are overloaded in handling all of it. On the other hand, involving expert knowledge in a lightweight way could provide significant gains in understanding the semantics of big data, with which accurate and interpretable highlights could be effectively refined.

VII. CONCLUSION

This work identifies and focuses on relieving the massive alert pitfall from the practice of production supercomputer systems. Through an exploratory study, we find that supercomputer alerts are a stream of bursts, costing operators several hours to handle, even after suppressing many empirically negligible ones. As a technical remedy, a novel SuperAgg framework is proposed based on exploring hierarchical patterns in supercomputer alerts. It successfully discovers 4 categories of single-channel alert patterns leveraging unsupervised learning and human-in-the-loop assistance and builds system-wise alert correlation rules with primary-and-secondary analysis. Dedicated aggregation strategies are proposed for the elaborated patterns, which yield interpretable alert suppression in the context of supercomputer alert reporting. Real-world evaluation and implementation demonstrate the effectiveness of the proposal in the production system.

ACKNOWLEDGMENT

This work is supported by the Science and Technology Innovation Program of Hunan Province under Grant 2023RC3027, the Advanced Research Project of China under Grant 31511010501, and the National Natural Science Foundation of China under Grant 62102425.

REFERENCES

[1] R. Wang, K. Lu, C. Juan, W. zhe Zhang, J. wen Li, Y. Yuan, P. jing Lu, L. Huang, S. Li, and X. Fan, "Brief introduction of tianhe exascale prototype system," *Tsinghua Science & Technology*, vol. 26, pp. 361–369, 2021.

[2] Q. Zhu, H. Luo, C. Yang, M. Ding, W. Yin, and X. Yuan, "Enabling and scaling the hpcg benchmark on the newest generation sunway supercomputer with 42 million heterogeneous cores," in *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–13.

[3] Y. Dai, Y. Dong, K. Lu, R. Wang, W. Zhang, J. Chen, M. Shao, and Z. Wang, "Towards scalable resource management for supercomputers," in *Proc. of International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2022, pp. 1–15.

[4] W. Shin, V. Oles, A. M. Karimi, J. A. Ellis, and F. Wang, "Revealing power, energy and thermal dynamics of a 200pf pre-exascale supercomputer," in *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–18.

[5] N. Zhao, J. Chen, X. Peng, H. Wang, X. Wu, Y. Zhang, Z. Chen, X. Zheng, X. Nie, G. Wang, Y. Wu, F. Zhou, W. Zhang, K. Sui, and D. Pei, "Understanding and handling alert storm for online service systems," in *Proc. of IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2020, pp. 262–263.

[6] D. Lin, R. Raghu, V. Ramamurthy, J. Yu, R. Radhakrishnan, and J. Fernandez, "Unveiling clusters of events for alert and incident management in large-scale enterprise it," in *Proc. of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (SIGKDD)*, 2014.

[7] J. Chen, P. Wang, and W. Wang, "Online summarizing alerts through semantic and behavior information," in *Proc. of IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 1646–1657.

[8] J. Xu, Y. Wang, P. Chen, and P. Wang, "Lightweight and adaptive service api performance monitoring in highly dynamic cloud environment," in *Proc. of IEEE International Conference on Services Computing (SCC)*, 2017, pp. 35–43.

[9] G. Werner, S. J. Yang, and K. McConky, "Near real-time intrusion alert aggregation using concept-based learning," in *Proc. of the 18th ACM International Conference on Computing Frontiers (CF)*, 2021.

[10] M. Landauer, F. Skopik, M. Wurzenberger, and A. Rauber, "Dealing with security alert flooding: Using machine learning for domain-independent alert aggregation," *ACM Transactions on Privacy and Security (TOPS)*, vol. 25, pp. 1 – 36, 2022.

[11] S. Salah, G. Maciá-Fernández, and J. E. D. Verdejo, "A model-based survey of alert correlation techniques," *Comput. Networks*, vol. 57, pp. 1289–1317, 2013.

[12] X. Tao, F. Jia, Y. Yu, D. Ding, Z. Cui, and L. Shi, "An intrusion alarm data association analysis method," in *Proc. of IEEE 19th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, 2022, pp. 99–107.

[13] G. Jiang, H. Chen, K. Yoshihira, and A. Saxena, "Ranking the importance of alerts for problem determination in large computer systems," in *Proc. of International Conference on Automation and Computing*, 2009.

[14] N. Zhao, P. Jin, L. Wang, X. Yang, R. Liu, W. Zhang, K. Sui, and D. Pei, "Automatically and adaptively identifying severe alerts for online service systems," in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2020, pp. 2420–2429.

[15] M. Zhang, P. Wang, and W. Wang, "Efficient consensus motif discovery of all lengths in multiple time series," in *Proc. of International Conference on Database Systems for Advanced Applications (DASFAA)*, 2022.

[16] X. Wang, K. Yin, Q. Ouyang, X. Wen, S. Zhang, W. Zhang, L. Cao, J. Han, X. Jin, and D. Pei, "Identifying erroneous software changes through self-supervised contrastive learning on time series data," in *Proc. of IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, 2022, pp. 366–377.

[17] P. Schäfer, A. Ermshaus, and U. Leser, "Clasp-time series segmentation," in *Proc. of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, 2021.

[18] C. Wang, K. Wu, T. Zhou, and Z. Cai, "Time2state: An unsupervised framework for inferring the latent states in time series data," in *Proc. of the ACM on Management of Data (SIGMOD)*, vol. 1, 2023, pp. 1 – 18.

[19] "A Linux Foundation Project open-source Baseboard Management Controllers (BMC) Firmware Stack," https://github.com/openbmc.

[20] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," 1993.

[21] L. Tang, T. Li, F. Pinel, L. Shwartz, and G. Grabarnik, "Optimizing system monitoring configurations for non-actionable alerts," *IEEE Network Operations and Management Symposium*, pp. 34–42, 2012.

[22] K. Madnani. (2018) The next generation of azure alerts has arrived. [Online]. Available: https://azure.microsoft.com/en-us/blog/the-next-generation-of-azure-alerts-has-arrived/