DEAR: Distributed Evaluation of Alerting Rules

Mathias Mormul, Pascal Hirmer, Christoph Stach, and Bernhard Mitschang
University of Stuttgart
Institute for Parallel and Distributed Systems
Universitätsstraße 38, 70569 Stuttgart
Email: firstname.lastname@ipvs.uni-stuttgart.de

***Abstract*—Cloud computing passed the hype cycle long ago and firmly established itself as a future technology since then. However, to utilize the cloud as cost-efficiently as possible, a continuous monitoring is key to prevent an over- or under-commissioning of resources. In large-scaled scenarios, several challenges for cloud monitoring, such as high network traffic volume, low accuracy of monitoring data, and high time-to-insight, require new approaches in IT Operations while considering administrative complexity. To handle these challenges, we present *DEAR*, the Distributed Evaluation of Alerting Rules. DEAR is a plugin for monitoring systems which automatically distributes alerting rules to the monitored resources to solve the trade-off between high accuracy and low network traffic volume without administrative overhead. We evaluate our approach against requirements of today's IT monitoring and compare it to conventional agent-based monitoring approaches.**

***Index Terms*—Cloud Monitoring; Agent-based; Alerting.**

I. Introduction

Cloud computing passed the hype cycle long ago and firmly established itself as a future technology since then. As resource utilization may vary over time, a continuous monitoring of cloud resources and especially an alerting system that informs system administrators about problems to enable them to resolve emerging issues are essential [1]. Furthermore, recently, the term AIOps, coined by Gartner [2], introduced the usage of machine learning to the domain of IT Operations to further optimize monitoring, e. g., dynamic alerting thresholds based on predictive analysis.

Usually, the collected monitoring data are stored on a centralized monitoring server [1], [3] and in scenarios that involve hundreds or thousands of virtual machines (VMs), each with multiple different applications, cloud monitoring can be classified as a Big Data problem [4]. Especially the volume of the monitoring data quickly increases to an unmanageable size for the network bandwidth and disk space [1], [5]. The common solution to this problem is the use of monitoring agents, which aggregates monitoring data before sending it to the monitoring server [5]. However, aggregation always leads to a loss of accuracy [6]. Aceto et al. [5] define a monitoring system as *accurate* when the measures provided are as close as possible to the real value. Therefore, an accurate and fine-grained monitoring is, in general, not possible by aggregating monitoring data. Especially the quality of alerting suffers from poor accuracy and the amount of false-negatives and false-positives leads to unnecessary remediation efforts or even to overlooking of serious problems.

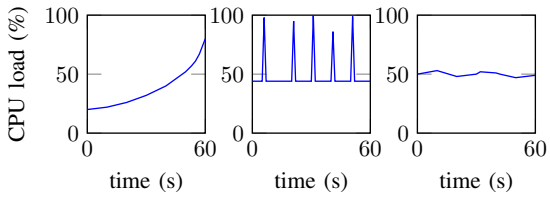

Figure 1. Exemplary CPU Loads of a Cloud Application.

As a consequence, decentralized monitoring architectures were created. On each VM, fully operational monitoring systems are installed that communicate with each other. Therefore, the monitoring data can be analyzed without generating network traffic volume while keeping a high accuracy. However, decentralization always leads to an increase in management complexity, as now several servers need to be managed instead of a single one [7]. Furthermore, as all monitoring data are spread across several servers, holistic analysis is cumbersome or not possible at all.

Our goal is a hybrid approach combining the advantages of centralization and decentralization to enable fine-grained alerting without the disadvantages of aggregation of monitoring data or decentralized monitoring architectures. For this, we introduce *DEAR*, a plugin for monitoring systems enabling **D**istributed **E**valuation of **A**lerting **R**ules. DEAR connects to the alerting framework of a monitoring system and distributes the alerting rules to the respective VMs while considering requirements regarding accuracy, network traffic volume, time-to-insight, and others. Required changes to the monitoring agent and the alerting framework are performed automatically and the management of alerting rules is kept centralized to maintain management complexity of monitoring systems. We evaluate the performance of DEAR and compare our approach to centralized, agent-based monitoring approaches.

The remainder of this paper is structured as follows: Section II introduces challenges towards cloud monitoring derived from literature. Section III discusses how related work handles these challenges. In Section IV, we present DEAR and its components. In Section V, we evaluate DEAR. Lastly, Section VI concludes this paper and gives an outlook to future work.

II. Challenges

In this section, we show the disadvantages of aggregation and the resulting inaccuracies based on the examples shown in Fig. 1. Exemplary, the graphs show the CPU usage in percent of an arbitrary cloud service over 60 seconds. The monitoring agent samples the CPU usage every second and calculates the mean average over 60 seconds[1] to reduce network traffic volume. In Fig. 1 (left), the CPU load is rising exponentially. However, the calculated average is below 50 % when in fact the CPU load is already at 80 % and probably still rising. This potential problem would only be detected after the next time window, i. e., in total after two minutes, when the new average suddenly is above 80 % or higher. In Fig. 1 (middle), multiple spikes might indicate a problem with the service. However, when aggregating the monitoring data, information about the spikes are lost. In comparison to Fig. 1 (left), this problem might not be detected at all as the average might not change in the future. In Fig. 1 (right), a static workload is shown, which is the only one that is accurately displayed by the calculated average value as each sample value is close to it. Based on those graphs, we derive challenges $C1 - C4$ and a universal challenge in cloud monitoring $C5$:

$C1$: **Accuracy:** Having fine-grained monitoring data is essential for a reliable monitoring of an IT environment to increase accuracy [5]. For example, only then, in Fig. 1 (middle), the spikes can be detected and responded to if required.

$C2$: **Big Data:** Fine-grained monitoring data are gained by frequent sampling. However, for multiple metrics across multiple VMs, this approach leads to a large amount of monitoring data which burdens network bandwidth and disc space [4]. Therefore, network traffic volume needs to be reduced while not negatively affecting accuracy.

$C3$: **Time-to-insight:** Reducing time-to-insight (TTI), i. e., the time between the occurrence of an event and an appropriate response to it, is mandatory to enable fast reactions [5]. In Fig. 1 (left), the earlier the increase in CPU load is detected, the earlier it can be responded to. In Fig. 1 (middle), the spikes can be detected and responded to if required.

$C4$: **Historical data:** Keeping records of past monitoring data is a must for analysis purposes [2]. In Fig. 1 (left), predictions can be made to preemptively react to potential problems before they arise, e. g., overloaded CPU. In Fig. 1 (right), the behavior of the application can be *learned* to improve alerting rules by dynamically narrowing upper and lower thresholds.

$C5$: **Administration:** IT departments are oftentimes understaffed [8]. Therefore, new approaches need to be automated to not further burden system administrators and support scalability [2]. Furthermore, in contrast to decentralized solutions, a Single Point of Administration must be maintained to ease the management of the monitored IT environment.

[1]Other forms of aggregation (MAX, MIN, etc.) lead to similar problems.

III. Related Work

In the following, we discuss related work regarding different monitoring architectures and their inadequacies in regard to the above-mentioned challenges. In **centralized monitoring architectures**, to reduce network traffic volume, the common practice is the aggregation of monitoring data at the monitoring agent [5]. Different aggregation strategies are considered [9], [10]. However, aggregation always leads to a decrease in accuracy [6], [11]. A similar approach to reduce the monitoring data volume is to simply reduce the sampling frequency, in which monitoring data are gathered. However, whereas sampling frequencies at 30 or more seconds were acceptable in the past, nowadays a fast reaction to undesirable states is obligatory [7], [12].

Therefore, n-tier and fully **decentralized monitoring architectures** exist. N-tier architectures consist of a hierarchical tree structure with the central server as the root and the monitored resources as the leaves. In between are nodes (for $n \geq 3$), which are fully operative monitoring systems that act as *central* servers for a subset of the monitored resources. All those intermediate monitoring systems send their results to the actual centralized server that represents the access point for the administrator or yet to another tier of monitoring systems. In fully decentralized monitoring architectures, there is no centralized component, i. e., each agent comprises a fully functional monitoring system, which communicates with other agents in the monitored environment. However, decentralization is always accompanied by an increased management complexity, which oftentimes is not acceptable [7].

Hybrid monitoring architectures aim at outsourcing parts or functions of the monitoring system close to the data source while maintaining a centralized component. For instance, Wang et al. introduce *Monalytics* [13], [14] to perform analytics close to the data source by defining topologies for a monitored environment. Physical systems are partitioned into *zones* and in each zone monitoring brokers receive collected data from agents and aggregate and analyze the data to reduce TTI and monitoring data volume. However, only lightweight analysis can be performed and higher-level analysis must be programmed explicitly. Furthermore, historical data is not considered and future analysis is not possible.

Hauser et al. [1] introduce the generation of resource profiles at the data source. For this, they develop a tool that runs on the physical host, collects data, and produces and updates a statistical representation of the resource utilization using, e. g., histograms and Markov chains. Instead of the raw monitoring data, the resulting resource utilization model is retrieved by tools for alerting and only when needed to reduce network traffic and disc space. Still, monitoring data are aggregated and accuracy is reduced. The authors claim that for alerting a lesser accuracy is acceptable, however, neither do they present any evidence to support this claim nor do they present the loss in accuracy compared to traditional monitoring approaches.

Shao et al. [15] introduce a runtime model based monitoring approach. Monitoring data are checked against predefined rules

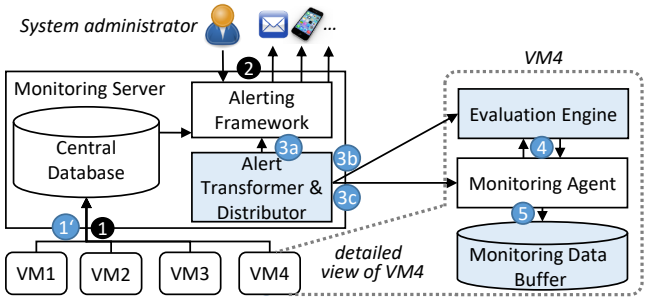

Figure 2. Integration of DEAR Components into Current Monitoring Architectures and Processes (steps introduced by DEAR depicted in blue).

on the monitoring agents to perform alerts if necessary. Similar to this, Moogsoft [16] introduces *Observe* data collectors to analyze data directly at the data source. The presented advantages are increased scalability due to less network traffic and an immediate insight due to lower latency. In both cases, those rules are managed directly on the monitoring agents, which leads to high management complexity in large-scaled environments.

Lastly, the distribution of alerting rules is related to distributed Complex Event Processing (CEP). Related works in this domain (e. g., [17], [18]) aim at distributing the processing of globally defined CEP queries close to the data sources to reduce network traffic and latency. We can use these approaches to distribute the condition part of our alerting rules. However, our scope goes beyond these approaches through alerting rule management, recording historical data, and the embedding of said approaches into the monitoring landscape such as automatic adaptations to monitoring agents and alerting frameworks.

IV. DEAR

With DEAR, we aim at providing a hybrid approach as a plugin for today's centralized monitoring systems to acquire the advantages of decentralized solutions, such as high accuracy, low TTI and low network traffic volume without adopting their increased management complexity. For this, DEAR needs to blend in with the current monitoring architecture. In Fig. 2, an extended centralized monitoring architecture with embedded DEAR components is shown as well as five process steps of monitoring. Exemplary, four VMs are monitored and their monitoring data are sent to a monitoring server. Marked with numbers (1) and (2) are the steps of agent-based monitoring and, in addition, numbers (3), (4), and (5) mark new steps from DEAR, which are briefly explained in the following.

(1): On each VM, a monitoring agent is installed as shown in the detailed view of *VM4* on the right in Fig. 2. For each monitoring agent, configurations include what needs to be monitored, e. g., CPU and RAM load, the sampling frequency f_s, and the aggregation interval I_a that defines for how long monitoring data are aggregated before they are sent to the monitoring server. Therefore, using aggregation, the network traffic volume is reduced by $(1 - \frac{f_s}{I_a}) * 100\%$ in comparison to no aggregation used at all.

(2): Based on those monitoring data, the system administrator defines alerting rules in an alerting framework. The alerting framework accesses all incoming data from the central database and continuously evaluates all alerting rules. If an alerting rule is violated, alerts are sent via different communication channels, such as email or text message to the personnel responsible. It becomes clear that alerting is heavily impacted by the aggregation interval I_a in regard to accuracy and TTI as aggregation leads to more coarse-grained data and monitoring data are sent at a later point in time, i. e., after the aggregation interval passed.

(3): Therefore, the component *Alert Transformer & Distributor* is introduced which accesses the alerting framework and transforms the alerting rules to a common query language so that they can be evaluated within the *Evaluation Engine* on the VM (3a). The transformed alerting rules are placed on the evaluation engine (3b) and the configuration of the monitoring agent is changed accordingly (3c) so that monitoring data are sent to the evaluation engine.

(4): Now, the monitoring agent sends the fine-grained monitoring data directly to the evaluation engine where the alerting rules are evaluated. The results of the evaluation – the alert data – are sent back to the monitoring agent which sends them to the monitoring server. At the same time, coarse-grained monitoring data are sent to the central database to maintain historical data.

(5): Furthermore, the monitoring agent sends fine-grained monitoring data to a monitoring data buffer where they are stored for a short amount of time so that, in case of alerts, a fine-grained analysis of recent monitoring data is possible, e. g., to perform a root cause analysis.

(1'): As a result of the previous steps, the communication between the monitoring agent and the monitoring server has changed, and step (1) is replaced by step (1'). Aggregated monitoring data are sent regularly to the monitoring server to maintain historical data without stressing the network. In case of alerts, relevant monitoring data from the monitoring data buffer are sent to the central database.

A. Alert Transformer & Distributor

The alert transformer and distributor is the central component of DEAR and is responsible for the transformation of alerting rules, distribution of said alerting rules to local evaluation engines, and adaptations of monitoring agents. The modular architecture supports different frameworks, agents, and evaluation engines by the creation of adapters as shown in Fig. 3 and explained in the following.

1) Alerting Framework Adapter: The alerting framework adapter is used to access alerting rules from an alerting framework. Based on the used alerting framework, the definition

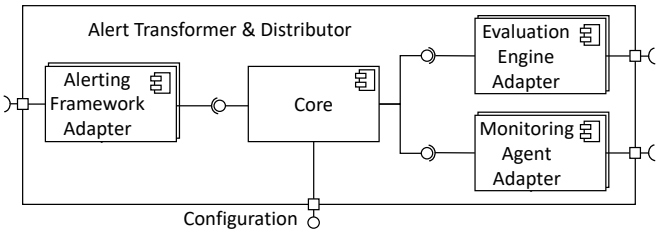

Figure 3. Components of Alert Transformer & Distributor.

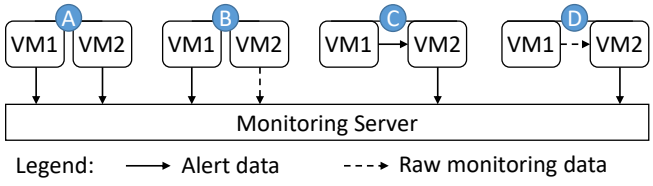

Figure 5. Possible Distributed Processing Strategies.

and attributes of an alerting rule differ. However, a condition and an action is always required. Furthermore, optional arguments, such as the name and severity of the alert, help system administrator to quickly classify an alert. As syntax and expressiveness of alerting rules differ from framework to framework, for each alerting framework, a new adapter needs to be programmed. First, the rule condition of the alerting rule is transformed into a uniform representation to reduce the amount of required transformations when supporting n alerting frameworks and m evaluation engines from $n * m$ to $n + m$ transformations. As uniform representation, we use binary expression trees (BETs)[2] [19] with inorder traversal where the nodes represent logical operators and the leaves represent conditions. A rule condition involving multiple conditions and its equivalent BET are shown in Fig. 4. The BET is used as input for the evaluation engine adapter where it is transformed into the according query language, e. g., a CEP query if a CEP engine is used. The original rule in the alerting framework is replaced by a placeholder that is also used for the calculated result in the evaluation engine. The condition rule in Fig. 4 is changed to $IF\ (VM1.A1)\ AND\ (VM2.A2)$ and the evaluation engine on VM1 calculates $(IF\ CPU > 90\%\ OR\ RAM < 200MB)\ THEN\ (A1 = true)$ and returns $A1$. Analogous, the evaluation engine on VM2 calculates and returns $A2$.

The rest of the alerting rule remains the same so the management of alerting rules, e. g., the different communication channels, such as email, and their configurations, can be managed globally by the system administrator as before.

2) Core: The core component is responsible for the distribution of alerting rules based on the BET. Using inorder traversal, the longest BET subtrees are detected whose leaves contain conditions for monitoring data from the same VM. Those subtrees are transformed into the query language of the evaluation engines by the evaluation engine adapters and distributed to the VMs. Now, the monitoring agent needs to send fine-grained monitoring data to the evaluation engine instead of the monitoring server. Therefore, the configuration of the monitoring agent is changed by the monitoring agent adapter to reroute monitoring data accordingly.

[2]Other representations might be feasible as well.

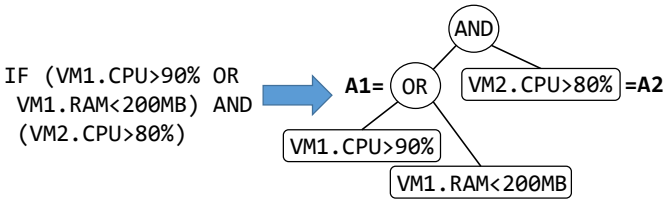

Figure 4. Transformation of Alerting Rule into Binary Expression Tree.

Based on the sources of the monitoring data of the rule condition, different distributed processing strategies can be applied as shown in Fig. 5. To illustrate the strategies, we use the rule condition shown in Fig. 4.

(A): All VMs evaluate their respective BET subtrees and send their result, i. e., the alert data, to the monitoring server, which represents the example shown in Fig. 4.

(B): If for any reason a VM cannot perform a local evaluation, only part of the condition rule is evaluated on the VMs and VM2 sends its monitoring data to the monitoring server.

(C): It is only of interest if both BET subtrees are evaluated to true since the highest level node of the BET is the *AND*-operator. Therefore, if only one of the subtrees is evaluated to true, alert data is sent to the monitoring server and no alert is created. This creates unnecessary network traffic and can be avoided by this strategy. One of the VMs sends its alert data to the other VM instead of the monitoring server, so that the complete BET can be evaluated before sending alert data to the monitoring server. However, this strategy only makes sense if the communication between the VMs is cheap, e. g., if they belong to the same cluster. Otherwise, e.g., in multi-cloud environments, strategy A is preferable since no unnecessary stress is put on the VMs.

(D): If one VM cannot perform a local evaluation and the communication to the other VM is cheap, raw monitoring data can be sent between the VMs so that the complete BET can be evaluated.

Analogous, the distributed processing strategies can be applied to alerting rules involving more than two VMs. Which strategy is best depends on the use case and the IT environment (e. g., VMs in the same cluster) and therefore, must be decided by the system administrator. For further optimization, distribution strategies from the domain of distributed CEP (see Section III) can be applied here.

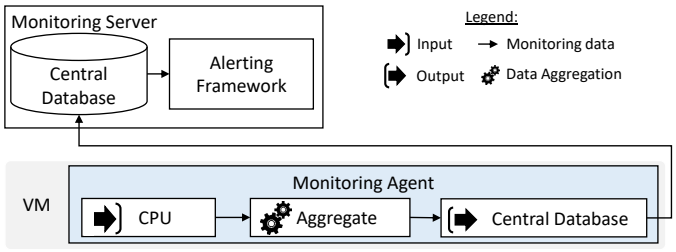

Figure 6. Original Agent Configuration.

3) Evaluation Engine Adapter: Different evaluation engines can be used, such as Esper[3]. For each, an evaluation engine adapter is required to transform BET trees into the corresponding query language of the evaluation engine.

4) Monitoring Agent Adapter: The monitoring agent adapter performs the required changes to the configuration of monitoring agents regarding the routing of monitoring data, i. e., it transforms a monitoring agent as shown in Fig. 6 into a monitoring agent shown in Fig. 7. Mormul et al. [20] introduced a generic model for agents that describes their architecture by an agent pipeline consisting of an input, processing, aggregation, and output node of which processing and aggregation are optional. In Fig. 6, the original monitoring agent is shown for the monitoring of CPU. The monitoring agent consists of an input to collect CPU measures in a predefined sampling frequency f_s, e. g., once per second. The collected data are passed to an aggregator node which aggregates the CPU measures by a certain aggregation interval I_a, e. g., 60 seconds. Lastly, in the output node, the data sink is defined, i. e., the central database. In this example, by aggregating monitoring data, the network traffic is reduced by $(1-\frac{f_s}{I_a})*100 = 98.\overline{3}\%$. The aggregated, and therefore less accurate, monitoring data are sent to the monitoring server, stored in the central database and analyzed by the alerting framework.

In comparison, Fig. 7 shows the agent configuration after the transformation of alerting rules to the evaluation engine on the VM. The routing paths RP within the monitoring agent are annotated by $RP1 - RP4$. $RP3$ is used for the evaluation of alerting rules. Since the monitoring data does not leave the VM, aggregation is not needed to reduce network traffic volume. Instead, the CPU measures are sent directly to the evaluation engine where the alerting rule is evaluated. The evaluation returns a boolean value $A1$ describing if the threshold of the alerting rule was exceeded. Only if the value transitions from $FALSE$ to $TRUE$ or from $TRUE$ to $FALSE$, the alert data is input for the monitoring agent in $RP1$ and sent to the central database.

It is clear that by evaluating the alerting rule locally, a detachment between sampling frequency f_s and network traffic volume is made, i. e., f_s can be increased to meet requirements of use cases that require fine-grained monitoring data without having concerns about straining the network.

[3]http://www.espertech.com/esper/

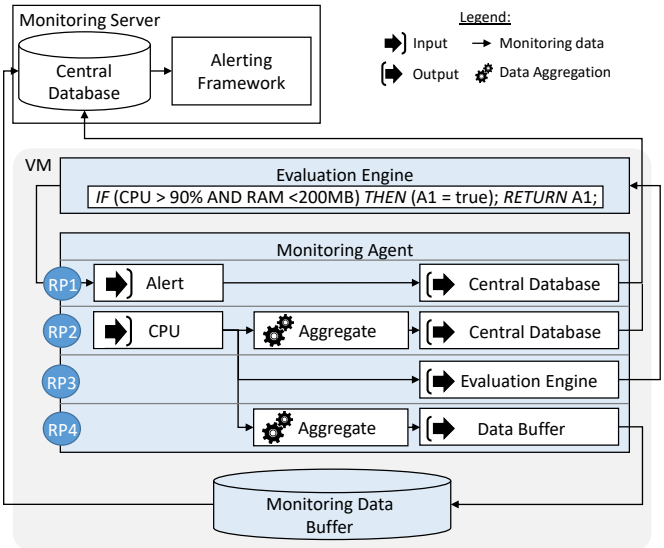

Figure 7. New Agent Configuration and DEAR Components.

$RP2$ represents the original routing path as shown in Fig. 6. Since the evaluation of the alerting rule now is performed locally, the aggregation interval does not affect the accuracy of monitoring data for alerting. Therefore, the aggregation interval can be increased to decrease network traffic and required disk space on the monitoring server.

Lastly, $RP4$ represents a routing path to enable fine-grained historic analysis in the case of alerts. All monitoring data are stored locally for a limited period. In case of a violated alerting rule, the stored monitoring data associated with the violated alerting rule are flushed to the central database on the monitoring server for later analysis, e. g., a root cause analysis. Due to disc space restrictions on the monitored VM, aggregation can be used to decrease monitoring data volume. However, this is only sensible if this aggregation interval I_{a_Buffer} is set lower than the aggregation interval I_a of $RP2$ since these monitoring data are sent to the monitoring server anyway.

$RP1$ and $RP3$ are mandatory to enable the evaluation of alerting rules on VMs. $RP2$ and $RP4$ are optional to enable the future analysis of coarse-grained monitoring data or root cause analysis on fine-grained monitoring data.

B. Evaluation Engine

The evaluation engine acts as a local alerting framework on the VMs. To handle the challenges presented in Section II, the engine needs to support low latency to enable low TTI, and high throughput in order to support Big Data. Furthermore, it needs to be light-weight in terms of memory, CPU and IO usage to minimize the additional stress on VMs, and expressive enough to express sophisticated alerting rules including temporal relationships from several alerting frameworks.

C. Monitoring Data Buffer

The monitoring data buffer acts as a short-term storage for monitoring data. The purpose is to enable a fine-grained analysis of the cause of an alert in addition to the continuous evaluation of alerting rules. For example, CPU load spikes in Fig. 1 could be symptomatic of an impending error. If this behavior is unknown, the aggregated monitoring data stored in the monitoring server does not allow the possibility to learn this relation. Using the monitoring data buffer, recent monitoring data is sent to the central database in case of an alert. These data can be used for historical analysis to detect the root cause of the alert and enable preventive measures or a faster reaction to an impending error. A further advantage of a monitoring data buffer is that, in case of a short failure of the central monitoring server, no monitoring data are lost.

To ensure compatibility between the monitoring data buffer and the central database and the interface between monitoring agent and monitoring data buffer, the easiest solution is to choose the same database for the monitoring data buffer as the central database.

V. Evaluation

In the following, first, we present the results of our evaluation. We select different configurations, based on which we show the advantages of DEAR. Then, we discuss the results and how the challenges introduced in Section II are solved.

A. Evaluation Results

In our experiments, we use two VMs hosted on an OpenStack managed private cloud[4] with the following characteristics for each VM: 2 VCPU, 4 GB RAM, 40 GB Disk, Ubuntu 16.04 Server. The monitoring system used for our prototype is the TIG-Stack (Telegraf[5], InfluxDB[6], Grafana[7]). Telegraf presents a plugin-based monitoring agent, InfluxDB a time-series database, and Grafana a web-based management interface with alerting capabilities. One VM is used as a monitoring server, whereas the other one acts as the monitored resource. We use the CEP engine Esper as the evaluation engine and InfluxDB as the monitoring data buffer. The rule conditions of the alerting rules are defined as shown in Listing 1.

```
A1: IF CPU.load > 90 %
A2: IF CPU.load > 90 % FOR 10s
A3: IF CPU.load > 90 % FOR 60s
```

Listing 1. Rule conditions used for evaluation

Exemplary, we monitor CPU loads of multiple VMs to check when a critical amount of VMs have their CPU load above 90%. For this, each VM evaluates alerting rules $A1 - A3$ and sends its results to the monitoring server where the occuring alerts can be counted. We test three different configurations and, for each, over one hour, we measure (i) the network traffic volume N between a VM and the monitoring server, (ii) the average CPU and RAM load of Telegraf and Esper, (iii) the TTI for $A1 - A3$, and (iv) the disc space on the monitoring data buffer. Lastly, we depict the signal of a generated time-series and its loss in accuracy for higher aggregation intervals. The evaluation results are shown in Table II and refer to configurations *Conf. 1*, *Conf. 2*, and *Conf. 3* shown in Table I.

Table I
Configurations for Evaluation.

Parameters	Conf. 1	Conf. 2	Conf. 3
Sampling frequency f_s	1 / s	1 / s	100 / s
Aggregation interval I_a	-	10s	60s
Aggregation interval I_{a_Buffer}	-	10s	-

Table II
Evaluation Results.

		Conf. 1	Conf. 2	Conf. 3
Network Traffic Volume (KB)		2042.52	203.01	34.12
Load (%) (Telegraf)	CPU	0.0	0.0	2.5
	RAM	1.1	1.1	1.3
Load (%) (Esper)	CPU	0.11	0.10	0.52
	RAM	0.61	0.67	1.07
Disc Space (KB)		300	72	27965
Time-to-insight (ms)	Traditional	28.8	6710.8	27211.1
	DEAR	379.4	360.9	377.6

The network traffic volume is measured between the VM and the monitoring server and is the same for the traditional approach and our approach since I_a is applied to both. As expected, the network traffic volume is reduced by increasing I_a.

The load of Telegraf and Esper did not change noticeably between *Conf. 1* and *Conf. 2*. However, in *Conf. 3*, sampling every 10 milliseconds results in a considerable overhead regarding CPU load.

Disc space refers to the allocated disc space of the monitoring data buffer. As expected, increasing I_{a_Buffer} reduces allocated disc space. Again, a very high sampling frequency in addition with no aggregation for the monitoring data buffer in *Conf. 3* leads to comparably large amount of monitoring data, approximately 28 MB.

To measure TTI, we create artificial monitoring data with a value starting at 0 % and instantly increasing it to 91 % at a random moment in time. We measure the time between the actual violations of the alerting rules and their detection by the alerting framework. To eliminate network delays, Telegraf and Esper are installed on the same VM as the monitoring server. We run alerting rule $A1$ for *Conf. 1*, $A2$ for *Conf. 2*, and $A3$ for *Conf. 3*. For each alerting rule, the experiment was repeated 20 times and the mean average was calculated. As shown in

[4]https://www.openstack.org/
[5]https://www.influxdata.com/time-series-platform/telegraf/
[6]https://www.influxdata.com/products/influxdb-overview/
[7]https://grafana.com/

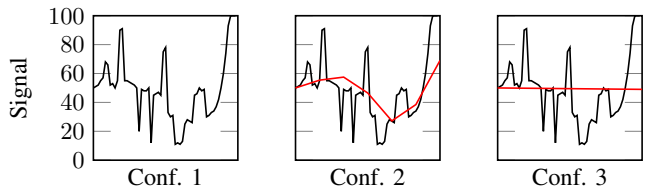

Figure 8. Original Signal (*black*) and Aggregated Signal (*red*) of an Arbitrary Time-Series of 60 Seconds.

Table II, the TTI for *Conf. 1* is higher than in the traditional approach due to an extended communication path. However, TTI remains similar across all configurations. In comparison, in the traditional approach, TTI is heavily impacted by I_a.

B. Discussion

In the following, we discuss if the challenges $C1 - C5$ introduced in Section II are solved based on the evaluation results and DEAR in general.

By distributing the evaluation of alerting rules close to the data source, we detached network traffic volume from granularity of monitoring data for alerting. Very fine-grained monitoring data are available for the evaluation of alerting rules ($C1$) without increasing network traffic volume ($C2$). To visualize the effect of aggregation and the resulting loss in accuracy, Fig. 8 shows the original (black) and aggregated (red) signal according to the aggregation intervals of *Conf. 1-3*. Using DEAR, the original signal is available in each configuration, whereas the traditional approach relies on the aggregated, less accurate signal in *Conf. 2-3*. Load of Telegraf and Esper is acceptable in configurations, such as *Conf. 2*, where there are already advantages over the traditional approach. However, CPU load is noticeable for high sampling frequencies. Therefore, such frequencies should be used with care and only when required.

When no aggregation (I_a) is used, TTI in DEAR is increased in comparison to the traditional approach. However, as we aimed at detaching network traffic volume from accuracy, it is not sensible to use DEAR without aggregation. When using aggregation (*Conf. 2 & Conf. 3)*, TTI in DEAR remains similar whereas TTI in the traditional approach is increased heavily based on the degree of aggregation. Therefore, when aggregation is used, TTI is decreased and challenge $C3$ is solved.

Still, aggregated monitoring data can be sent to the monitoring server (see $RP2$ in Fig. 7) to maintain historical data for analysis purposes. Those data can be used to analyze the general behavior of a VM where a comparably high aggregation interval can be set, e. g., to detect if one VM behaves significantly different than the rest. For long-term predictions, e. g., *the rise or fall of workload in the next hours or days*, fine-grained data may not be as important. For example, machine learning models may overfit the data if there are too many outliers and noise, which may even reduce the accuracy of the models and, therefore, the predictions. By aggregating monitoring data, fine-grained monitoring data becomes trend data. Noise and outliers in the data can be reduced. Therefore, predictive models may produce better predictions and, furthermore, system administrators gain a more intuitive overview and understand the system state at a glance. However, long aggregation intervals mean that VMs do not communicate with the monitoring server and failed VMs are not detected within this period. In this case, light-weight heartbeat messages are required to periodically check the responsiveness of VMs or push notifications from the cloud management system in case of failures. In addition, a monitoring data buffer stores most recent monitoring data for a limited period. In case of alerts, the monitoring data associated with the violated alerting rule are flushed to the central database. This way, on-demand fine-grained monitoring data can be used, e. g., to analyze root causes of alerts. However, selecting an appropriate retention for local data is important to not allocate too much disc space on the monitored VMs. In conclusion, historical data are maintained and $C4$ is solved.

Lastly, since only the rule conditions are distributed, the management of alerting rules is maintained centrally at the monitoring server and their management does not change. The distribution itself is fully automated and does not increase administration complexity ($C5$). Therefore, our goal to present a hybrid approach combining the advantages of both centralized and decentralized approaches without their disadvantages is reached. However, more configurable parameters exist and different distributed processing strategies are available. To support decision making for those configurations, metadata about the cloud environment, e. g., hypervisors and VMs, and others can be useful. For instance, Mormul et al. [21] introduce a framework for the holistic management of monitoring data and their metadata. These metadata can help decision making for the distributed processing strategies shown in Fig. 5, e. g., by providing the information if the VMs involved in the distribution process are in the same cloud.

VI. Summary and Future Work

The monitoring of complex cloud environments can lead to several challenges, such as high network traffic volume and high TTI. We analyze those challenges and introduce the hybrid approach DEAR as a plugin for agent-based monitoring systems to automatically distribute the evaluation of alerting rules to the data sources. We present the Alert Transformer & Distributor that accesses alerting rules from alerting frameworks, transforms them so that evaluation engines on the monitored resources can evaluate them, and adapts monitoring agents accordingly. Fine-grained monitoring data are sent to the evaluation engine whereas aggregated monitoring data and alert data are sent to the monitoring server. We evaluate our approach by measuring network traffic volume, load of the monitoring agent and evaluation engine, time-to-insight, and allocated disc space.

For future work, we plan to evaluate existing distributed processing strategies from the domain of distributed CEP and their application to our approach.

Acknowledgment

This work is partially funded by the BMWi project IC4F (01MA17008G).

References

[1] C. B. Hauser and S. Wesner, "Reviewing Cloud Monitoring: Towards Cloud Resource Profiling," in *Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing*, series CLOUD '18, IEEE, 2018, pages 678–685. DOI: 10.1109/CLOUD.2018.00093.

[2] S. Paskin. "What Is AIOps? A Beginner's Guide," BMC Software, Inc. (April 2020), [Online]. Available: https://www.bmc.com/blogs/what-is-aiops/.

[3] K. Alhamazani, R. Ranjan, K. Mitra, F. Rabhi, P. P. Jayaraman, S. U. Khan, A. Guabtni, and V. Bhatnagar, "An overview of the commercial cloud monitoring tools: Research dimensions, design issues, and state-of-the-art," *Computing*, volume 97, pages 357–377, 2015. DOI: 10.1007/s00607-014-0398-5.

[4] B. Harzog. "Modern monitoring is a big data problem," IDG Communications, Inc. (April 2017), [Online]. Available: https://www.networkworld.com/article/3191479/modern-monitoring-is-a-big-data-problem.html.

[5] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, volume 57, number 9, pages 2093–2115, 2013. DOI: 10.1016/j.comnet.2013.04.001.

[6] D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Monitoring Elastically Adaptive Multi-Cloud Services," *IEEE Transactions on Cloud Computing*, volume 6, number 3, pages 800–814, 2018. DOI: 10.1109/TCC.2015.2511760.

[7] J. S. Ward and A. Barker, "Observing the clouds: A survey and taxonomy of cloud monitoring," *Journal of Cloud Computing*, volume 3, number 24, pages 1–30, 2014. DOI: 10.1186/s13677-014-0024-2.

[8] J. Popovic. "All Things Are Digital In Business, But Finding Digital Talent Is A Tall Order," Robert Half Technology. (November 2017), [Online]. Available: https://rh-us.mediaroom.com/2017-11-01-All-Things-Are-Digital-In-Business-But-Finding-Digital-Talent-Is-A-Tall-Order.

[9] R. Mehrotra, A. Dubey, S. Abdelwahed, and W. Monceaux, "Large Scale Monitoring and Online Analysis in a Distributed Virtualized Environment," in *Proceedings of the 2011 Eighth IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems*, series EASE '11, IEEE, 2011, pages 1–9. DOI: 10.1109/EASe.2011.17.

[10] F. Azmandian, M. Moffie, J. G. Dy, J. A. Aslam, and D. R. Kaeli, "Workload Characterization at the Virtualization Layer," in *Proceedings of the 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, series MASCOTS '11, IEEE, 2011, pages 63–72. DOI: 10.1109/MASCOTS.2011.63.

[11] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, and V. Stankovski, "Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review," *Journal of Systems and Software*, volume 136, pages 19–38, 2018. DOI: 10.1016/j.jss.2017.10.033.

[12] A. Varghese and D. Tandur, "Wireless requirements and challenges in Industry 4.0," in *Proceedings of the 2014 International Conference on Contemporary Computing and Informatics*, series IC3I '14, IEEE, 2014, pages 634–638. DOI: 10.1109/IC3I.2014.7019732.

[13] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, and M. Wolf, "A Flexible Architecture Integrating Monitoring and Analytics for Managing Large-Scale Data Centers," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, series ICAC '11, ACM, 2011, pages 141–150. DOI: 10.1145/1998582.1998605.

[14] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf, "Monalytics: Online Monitoring and Analytics for Managing Large Scale Data Centers," in *Proceedings of the 7th International Conference on Autonomic Computing*, series ICAC '10, ACM, 2010, pages 141–150. DOI: 10.1145/1809049.1809073.

[15] J. Shao, H. Wei, Q. Wang, and H. Mei, "A Runtime Model Based Monitoring Approach for Cloud," in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, series CLOUD '10, IEEE, 2010, pages 313–320. DOI: 10.1109/CLOUD.2010.31.

[16] Moogsoft. "Moogsoft Observe." (2019), [Online]. Available: https://docs.moogsoft.com/AIOps.7.1.0/Moogsoft-Observe_29953305.html.

[17] N. P. Schultz-Møller, M. Migliavacca, and P. Pietzuch, "Distributed Complex Event Processing with Query Rewriting," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, series DEBS '09, ACM, 2009, pages 1–12. DOI: 10.1145/1619258.1619264.

[18] K. Sun, Y. Wang, and S. Peng, "Distributed complex event processing using rule deployment," in *Proceedings of the 2015 International Conference on Education, Management and Computing Technology*, series ICEMCT '15, Atlantis Press, 2015, pages 1244–1250. DOI: 10.2991/icemct-15.2015.255.

[19] B. R. Preiss, *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*, D. Barron and P. Wegner, Eds., series Worlswide Series in Computer Science. New York, Chichester, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons, Inc., 1998.

[20] M. Mormul, P. Hirmer, C. Stach, and B. Mitschang, "Avoiding Vendor-Lockin in Cloud Monitoring using Generic Agent Templates," in *Proceedings of the 23rd International Conference on Business Information Systems (BIS 2020)*, series LNBIP, volume 389, Springer, 2020, pages 367–378. DOI: 10.1007/978-3-030-53337-3_27.

[21] M. Mormul and C. Stach, "A Context Model for Holistic Monitoring and Management of Complex IT Environments," in *Proceedings of the 2020 IEEE International Conference on Pervasive Computing and Communications Workshops*, series CoMoRea '20, IEEE, 2020, 47:1–47:6. DOI: 10.1109/PerComWorkshops48775.2020.9156101.