Monitoring the distributed cloud: Metric collection and aggregation

Tamara Ranković[0000-0002-5265-4626], Nemanja Pokornić, Aleksandar Pavlović and Miloš Simić[0000-0001-8646-1569]

Faculty of Technical Sciences, University of Novi Sad, Serbia
tamara.rankovic@uns.ac.rs

Abstract. As new cloud models evolve, traditional monitoring solutions often struggle to meet the scalability, lightweight operation, and resilience requirements of modern systems. This paper introduces a hybrid monitoring system tailored for distributed cloud (DC) environments, which face the challenges of dynamic, geographically dispersed, and resource-constrained infrastructures. Our approach decentralizes both data collection and aggregation within regions, effectively reducing latency and avoiding the bottlenecks that would arise from centralizing these processes in the cloud. Aiming to minimize the overhead on individual nodes, the system offloads data collection and processing to metrics servers located within each region. Configurable filtering strategies ensure efficient data handling by discarding insignificant changes, while continuous aggregation at the region-level metrics servers improves query responsiveness. To enhance resilience, the system leverages cloud-managed replication and failure detection, ensuring uninterrupted service even during server or network failures. Real-time alerting is handled locally on metrics servers, reducing response times. The system supports aggregated views through client-defined aggregation rules, offering timely insights into system performance. Our solution is designed to scale efficiently as new nodes and regions are added, with minimal impact on network load and performance.

Keywords: metrics, monitoring, cloud, distributed cloud, observability.

1 Introduction

As cloud environments grow in scale and complexity, ensuring their reliability and performance becomes increasingly challenging [1]. This is especially true for novel cloud-extending paradigms, such as edge and fog computing [2, 3], where numerous, geographically distant, and potentially resource-limited components need to work together seamlessly. In such systems, effective monitoring is essential for maintaining operational stability, detecting issues early, and ensuring that resources are used efficiently.

At the heart of any monitoring system are the metrics that track the system's health and performance. These metrics are used to evaluate everything from the response time of services to the utilization of system resources. They are critical not only for human administrators but also for automated systems like workload schedulers, autoscalers,

and self-healing mechanisms, which rely on metrics to make real-time decisions. In addition, machine learning models can leverage historical metric data to improve decision-making or predict potential system failures before they happen.

Alongside these metrics, many systems incorporate alerting as a key monitoring strategy. Alerts are triggered when certain conditions, such as a threshold being exceeded, are met. This feature helps to provide immediate notifications about critical events, enabling quick corrective action or automatic responses to minimize downtime and system failure.

However, despite the wide availability of monitoring solutions, many of them are not built to accommodate the specific needs of emerging cloud models, including the distributed cloud (DC) model [4, 5]. These environments are inherently dynamic, with resources frequently being added, removed, or relocated. This constant change, combined with the need for multi-tenant isolation and resource-limited nodes, presents unique challenges for monitoring. The monitoring system must be flexible enough to adapt to frequent changes, while also being lightweight enough to avoid unnecessary overhead on the system.

This paper introduces a hybrid monitoring system designed specifically for DCs. The system is focused on collecting metrics from distributed nodes, aggregating the data, and serving it to a variety of clients. Our solution emphasizes modularity, efficiency, and scalability, ensuring it can seamlessly integrate with the flexible, often unpredictable nature of DC environments.

In Section 2 we provide an overview of existing monitoring solutions, focusing on those designed for distributed and cloud environments. Section 3 is dedicated to the in-depth exploration of the DC model, its main properties and requirements for the monitoring system extracted from these properties. In Section 4 we propose the architecture of the monitoring system for DCs, while in Section 5 we discuss how it satisfies the requirements. Section 6 concludes the paper and presents directions for future work.

2 Related work

Ganglia [6] is a distributed monitoring system tailored for high-performance computing environments, leveraging a scalable, decentralized architecture. Each node runs a gmond daemon that collects local metrics and disseminates them using multicast, allowing all nodes to maintain a full view of the cluster's state. Gmetad daemons periodically poll gmond instances or other gmetad daemons to aggregate metrics, forming a hierarchical tree structure that enables scalable data federation. Multiple gmetad daemons can be deployed per cluster to enhance fault tolerance. While effective for static infrastructures, Ganglia's static configuration model, lack of integrated alerting, and the overhead of storing full cluster state on each node limit its suitability for dynamic, cloud-native, or resource-constrained environments.

Prometheus[1] is a popular open-source monitoring system designed for dynamic and distributed environments. It uses a pull-based model where a central server scrapes

[1] Prometheus: https://prometheus.io/

metrics from instrumented targets, supporting service discovery to automatically track changing endpoints typical of edge deployments. Prometheus stores data in a time-series database and provides a powerful query language for flexible analysis and alerting. Its lightweight architecture and native support for containerized and ephemeral workloads make it well-suited for resource-constrained and highly dynamic edge environments. However, Prometheus's single-node design can limit scalability and high availability without additional tooling.

Netdata[2] uses a decentralized architecture where agents run on individual nodes, collecting metrics and evaluating alerting rules locally in real time. Parent nodes act as aggregation points, collecting streamed metrics from multiple agents to provide centralized visualization and long-term analysis. This design makes Netdata well-suited for high-fidelity, real-time data monitoring with rich and interactive visualization features. Metrics are primarily stored locally for a short time to preserve system resources, with long-term storage typically handled by integration with external databases. However, a key limitation is that alerting rules are not evaluated on aggregated data at the parent nodes, restricting alerts to the scope of individual agents rather than across the entire monitored infrastructure.

Acala [7] is a monitoring framework designed for federated Kubernetes clusters, providing a global view across multiple clusters. It employs a hierarchical architecture where a Global View Cluster hosts Prometheus storage and Acala controllers that periodically request metrics from member clusters. Each cluster runs an Acala member responsible for gathering node-level metrics, applying data reduction techniques such as aggregation, calculating cluster-wide averages, and deduplication, which filters out unchanged metric values before responding. These strategies reduce data volume and improve efficiency, enabling scalable and effective monitoring of federated Kubernetes environments.

EdgeCloud Mon [8] is a lightweight monitoring system designed for edge-cloud integration, targeting K3s[3] edge clusters. It collects infrastructure, pod, and virtual machine metrics using Prometheus and custom exporters. Each node runs a standard Node exporter[4] and a custom agent for detailed hardware metrics, while a KubeVirt exporter on the master node gathers metrics on virtual machines, supporting use cases like Windows-based workloads. Metrics are periodically scraped by a local Prometheus instance and pushed to a centralized cloud backend for long-term storage and analysis. While effective for small to medium scale deployments, EdgeCloud Mon may face challenges in large-scale, dynamic environments due to potential overhead from centralized metric aggregation.

3 Distributed clouds

As cloud computing continues to evolve, emerging architectures seek to address limitations inherent in traditional centralized cloud models, such as latency sensitivity, data

[2] Netdata: https://www.netdata.cloud/

[3] k3s: https://k3s.io/

[4] Node exporter: https://github.com/prometheus/node exporter/

privacy, and location-specific resource demands. One such architectural innovation is the DC model, which integrates centralized cloud infrastructure with dynamic, user-driven decentralized components. This hybrid model provides a more flexible and adaptive platform for managing heterogeneous and geographically dispersed workloads.

The DC model is structured in two primary layers: the cloud layer and the DC layer. The cloud layer represents the conventional centralized cloud infrastructure, which handles global orchestration and provides a control plane for the system. In contrast, the DC layer is composed of ephemeral, user-defined DCs, dynamically instantiated and terminated based on user needs. These DCs are specifically tailored in terms of resource capacity, geographic location, and workload specifications to meet strict latency and privacy requirements. The control plane in the cloud layer oversees the lifecycle of each DC, ensuring that it converges toward the user-specified configuration and operational goals. The system possesses a configuration subsystem that handles configuration validation and dissemination [9, 10].

A defining characteristic of the DC model is its hierarchical structure, which supports scalable and modular deployment. At the lowest level, each cluster comprises multiple nodes. Clusters are aggregated into regions, which are typically co-located to minimize communication delays and enhance cooperation. Multiple regions form a topology, enabling high-level orchestration tasks such as data replication, inter-region load balancing, and application failover. This multi-layered organization not only supports geographic and operational flexibility but also introduces opportunities for region offloading and high availability mechanisms across distributed zones.

To manage the complexities of resource sharing and ensure security in such a multi-tenant environment, the model employs hierarchical namespaces and dataspaces [11, 12], which provide robust isolation and enable fine-grained access control. Users interact with the system primarily through specification interfaces, where they define the configuration and desired behavior of their distributed clouds. The control plane then takes responsibility for provisioning infrastructure, orchestrating workloads, and maintaining compliance with user-defined constraints.

Properties of DCs most influential to our work are:

- ***Heterogeneity***: Clusters can contain nodes with diverse characteristics, many of them being commodity hardware with limited resources.
- ***Highly dynamic nature***: As users can form, alter and dispose infrastructure based on their potentially frequently changing needs, we expect it to continuously transform with many nodes joining or leaving at the same time.
- ***Large scale***: It is possible for a cluster to contain thousands or tens of thousands of nodes. This behavior is anticipated as the expected capabilities of individual nodes are low.
- ***Communication over the Internet***: No specialized network providing high speed or reliability is intended for use.

The metrics API should provide information both about individual nodes and entire clusters or regions. Clients should be able to see the current state, as well as the historical data in a specified time interval.

Requirements for the monitoring system we obtain from the DC properties and the API are the following:

- ***Scalable***: The system should be able to handle a large total number of nodes and a wide range of cluster sizes.
- ***Lightweight***: As nodes' resources are already limited, monitoring-related processes should consume as little as possible of them. Also, the traffic overhead shouldn't significantly overload the network or degrade performance of neither the control plane nor the nodes.
- ***Resilient***: The system should continue to operate even with high rates of node and network failures and frequent infrastructure changes present.
- ***Supports aggregated views***: As the API supports higher-level metrics that are generated from node-level metrics, the system must have an aggregation mechanism compatible with previously stated requirements.

4 Monitoring system architecture

In this section we present the design of a system for monitoring DCs that utilizes its properties and meets the requirements stated in Section 3. Our approach is hybrid in a sense that collection and aggregation are decentralized, performed inside regions and not the cloud layer, which is still relied-upon for coordination. The architecture of our solution is displayed in Fig. 1.

On every node there is a metrics client running, which collects metrics regarding hardware, operating system, isolated environments (virtual machines, containers, unikernels) in which client applications are running, and, if implemented, applications themselves. It does so by communicating with exporters that expose metrics from various sources. We require all exporters to implement the identical API, as it drastically reduces the complexity of the metrics client implementation. The API consists of only one endpoint that, when invoked, responds with a list of current values for all metrics of that exporter's target. The metrics client itself must also implement this API as it acts as the exporter for the entire node. Metrics should be served in the OpenMetrics[5] format, as it is a widely adopted cloud-native standard.

In every region there is a metrics server dedicated to collecting metrics from metrics clients of all nodes that are comprising clusters of that region. We opted for a metrics server per region because a region is a geographically concentrated unit, implying that intraregional latency is expected to be lower than interregional one. The metrics server periodically pulls data from metrics clients by invoking the exporter API. The client that received the request then gathers current metrics values from all its exporters, labels them so that their source can later be identified, and responds to the server.

The role of a metrics server is not only to collect data, but also to store it inside a time-series engine. If necessary, data can be transferred to the cloud, but only in cases when storage capacity is reached. To minimize the probability of this happening, we

[5] OpenMetrics: https://openmetrics.io/

enable the application of time-series filtering strategies. The first is delta-based filtering, which keeps the current data point only if it deviates from the last point more than a specified delta. The second is threshold-based filtering, which keeps only the points above or below the set thresholds. DC clients are the ones who specify what filtering strategies are applied inside a region, enabling them to strike a balance between data fidelity and resource utilization.

In addition to filtering, metrics servers are also responsible for performing aggregation. As aggregated views are one of the requirements, relying on on-demand aggregation is suboptimal, as it introduces additional computational overhead at query time. To mitigate this, we adopt a continuous aggregation approach, wherein aggregated metric values are computed at the time of data collection from individual nodes. Aggregation behavior is defined through rules specified by DC clients. Each rule comprises the name of the resulting aggregated series, a label selector that determines the set of input metrics, and an aggregation function. The label selector not only filters the relevant input series but also defines the label set for the output metric. For example, the rule *(cluster_mem_used, [cluster=c1], sum)* defines an aggregated metric cluster_mem_used, representing the total memory usage across all nodes within cluster c1.

Since metric servers represent potential single points of failure, we introduce replication and replacement strategies to improve system resiliency. These mechanisms are coordinated by the cloud layer, which is responsible for selecting both replicas and successors. To make informed decisions, the cloud layer requires visibility into the current state of all metric servers. For this purpose, each metric server periodically reports its own metrics to the cloud.

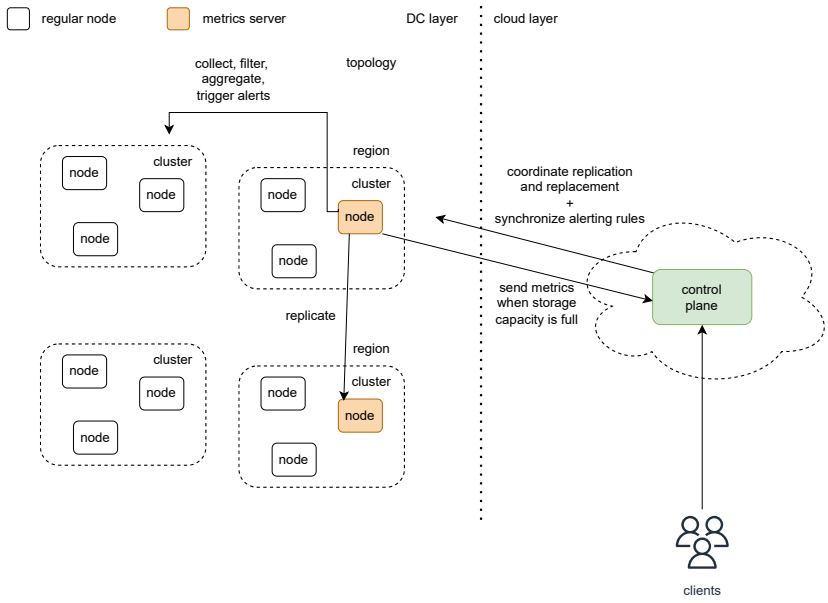

Fig. 1. Monitoring system architecture.

When a new region is initialized, one of the nodes is selected to serve as the initial metric server for that region. Given the heterogeneity in node resource capacities, candidates are limited to those with the highest available storage and compute power. Among the eligible candidates, one is chosen at random. The cloud then promotes the selected node by sending a notification that includes the collection interval (as specified by the DC client) and the replica set.

The replica set consists of metrics servers from other regions within the same topology, to which the newly designated server should forward the metrics it collects. The size of this set is determined by the replication factor, also configured by the DC client. A metrics server is selected as a replica based on its current replica load: preference is given to those serving as replicas for the fewest other servers, promoting an even distribution of data replication responsibilities across the system.

If the desired number of replicas is not available at the time the region is created, the cloud will incrementally update the replica set. As additional regions are brought online and new metric servers become available, the cloud notifies existing servers of new replicas, allowing the system to gradually reach the configured replication factor. Additionally, if replica failure is detected at any point in the future, the cloud performs the same process.

When a failure of an existing metrics server is detected, the cloud is responsible for promoting a new node to take over its role. The failure detection mechanism relies on the metrics server periodically sending its metrics to the cloud. If this communication does not occur within a predefined time window, the cloud initiates a verification process by contacting the replicas of that server to determine whether they have recently received any communication from it. If all replicas report no recent interaction, the metrics server is considered failed.

In response to a failure, a successor must be elected using the same procedure as during the initial selection of a metrics server. However, a key difference arises: during the creation of a new region, all nodes have equal total and available resources, which simplifies the ranking and selection process. In contrast, for an existing region with active workloads, the cloud must consider the current state of each node. This state information resides within the DC layer and cannot be retrieved from the failed metrics server. Instead, the cloud queries one of its replicas to obtain this information. Once retrieved, the remainder of the selection process proceeds as in the initial setup.

If the previously failed metrics server becomes available again, it is initially unaware that it has been demoted and may resume attempting to collect data from nodes. To prevent redundant metrics collection, each request sent by a metrics server includes a timestamp indicating when that server was elected. If a node receives a request with an older timestamp than one it has already processed, it will inform the sender that a newer metrics server is in place and that it should cease performing that role.

Collected metrics can be accessed by clients through queries or used to trigger alerts when specific conditions are met. The cloud control plane serves as the entry point for submitting such queries. Upon receiving a request, it determines whether the relevant data resides in the cloud layer or the DC layer and routes the request accordingly.

For alerting, we leverage data locality by executing alerting rules directly on the metrics servers where the data is stored. This approach minimizes latency and enables

immediate response actions. Similar to queries, the control plane also acts as the entry point for defining alerting rules. It is responsible for synchronizing these rules with the appropriate metrics server, which evaluates them during each data collection cycle.

A potential challenge arises from the fact that the most recent metrics data is stored within the DC, while some cloud-based components, such as schedulers or autoscalers, may require access to this data to make real-time decisions. To address this, each metrics server exposes an API that provides access to the latest core metrics, ensuring that critical cloud services can retrieve up-to-date information when needed.

5 Discussion

The proposed monitoring system meets scalability requirements by decentralizing metrics collection and aggregation within regions, while relying on the cloud layer for coordination. This regional focus reduces latency and network load, enabling the system to scale smoothly as new nodes and regions are added without centralized bottlenecks.

Lightweight operation is ensured through a simple, uniform API implemented by all exporters and clients, minimizing overhead on individual nodes. Configurable filtering strategies reduce data volume by discarding insignificant metric changes. Continuous aggregation at metrics servers precomputes summaries during collection, avoiding costly on-demand aggregation and improving query responsiveness.

Resilience is achieved via cloud-managed replication and failure detection. The cloud monitors metrics servers through periodic reports and verifies failures by consulting replicas. Upon detecting failure, the cloud promotes suitable successors based on current resource availability, maintaining uninterrupted service.

The system supports aggregated views by applying client-defined aggregation rules continuously during data collection. This approach produces timely, customizable summaries without incurring high query-time overhead, enabling efficient insight into system performance across nodes and regions.

6 Conclusion

This paper presents a hybrid monitoring system designed for DCs, addressing the unique challenges of monitoring dynamic, resource-constrained, and geographically dispersed environments. By decentralizing data collection and aggregation within regions, the system reduces latency and avoids bottlenecks typical of centralized systems. The use of a lightweight API and configurable filtering strategies ensures minimal overhead on nodes, while continuous aggregation improves query performance.

The system achieves resilience through replication and failure detection mechanisms managed by the cloud layer, ensuring uninterrupted service even in the face of failures. Real-time alerting is handled locally on metrics servers, reducing response times.

Future work will focus on two main directions: First, we aim to further decentralize the system by removing the need for dedicated metrics servers and reducing reliance on the control plane for coordination. Second, we plan to extend the monitoring system

to handle additional data types, such as logs and traces, while also incorporating support for probing and other advanced monitoring techniques.

Acknowledgements. Funded by the European Union (TaRDIS, 101093006). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

1. B. Costa, J. Bachiega Jr, L. R. Carvalho, M. Rosa, and A. Araujo, "Monitoring fog computing: A review, taxonomy and open challenges," Computer Networks, vol. 215, p. 109189, 2022.
2. W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE internet of things journal, vol. 3, no. 5, pp. 637–646, 2016.
3. F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in Proceedings of the first edition of the MCC workshop on Mobile cloud computing, 2012, pp. 13–16.
4. M. Simić, I. Prokić, J. Dedeić, G. Sladić, and B. Milosavljević, "Towards edge computing as a service: Dynamic formation of the micro data- centers," IEEE Access, vol. 9, pp. 114 468–114 484, 2021.
5. M. Simić, G. Sladić, M. Zarić, and B. Markoski, "Infrastructure as software in micro clouds at the edge," Sensors, vol. 21, no. 21, p. 7001, 2021.
6. M. L. Massie, B. N. Chun, and D. E. Culler, 'The ganglia distributed monitoring system: design, implementation, and experience', Parallel Computing, vol. 30, no. 7, pp. 817–840, 2004.
7. C.-K. Huang and G. Pierre, "Aggregate monitoring for geo-distributed kubernetes cluster federations," IEEE Transactions on Cloud Computing, 2024.
8. I. Korontanis, A. Makris, and K. Tserpes, "Edgecloud mon: A lightweight monitoring stack for k3s clusters," SoftwareX, vol. 26, p. 101675, 2024.
9. T. Ranković, I. Kovačević, V. Maksimović, G. Sladić, and M. Simić, "Configuration management in the distributed cloud," in Conference on Information Technology and its Applications. Springer, 2024, pp. 224– 235.
10. T. Ranković, F. Šiljić, J. Tomić, G. Sladić, and M. Simić, "Misconfiguration prevention and error cause detection for distributed-cloud applications," in 2024 IEEE 22nd Jubilee International Symposium on Intelligent Systems and Informatics (SISY). IEEE, 2024, pp. 000 297– 000 302.
11. M. Simić, J. Dedeić, M. Stojkov, and I. Prokić, "A hierarchical namespace approach for multi-tenancy in distributed clouds," IEEE Access, 2024.
12. M. Simić, J. Dedeić, M. Stojkov, and I. Prokić, 'Data overlay mesh in distributed clouds allowing collaborative applications', IEEE Access, 2025.