2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC) | 978-1-6654-6087-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/UCC56403.2022.00026

DEMon: Decentralized Monitoring for Highly Volatile Edge Environments

Shashikant Ilager, Jakob Fahringer, Samuel Carlos de Lima Dias, Ivona Brandic
High Performance Computing Systems Research Group
Vienna University of Technology, Austria

***Abstract*—Monitoring systems play an essential role in efficiently managing resources and application workloads by collecting, storing, and providing requisite information about the state of the resources. However, traditional monitoring systems that collect and store the data in centralized remote servers are infeasible for Edge environments. The centralized architecture increases the communication latency for information storage and retrieval and creates a failure bottleneck. In addition, the Edge resources are arbitrarily (de)provisioned, which creates further challenges for providing quick and trustworthy data. Thus, it is crucial to design and build a monitoring system that is fast, reliable, and trustworthy for such volatile Edge computing systems. Therefore, we propose a Decentralised Edge Monitoring (DEMon) framework, a decentralized, self-adaptive monitoring for highly volatile Edge environments. DEMon, at the core, leverages the stochastic Gossip communication protocol and develops techniques for efficient information dissemination, communication, and retrieval, avoiding a single point of failure and ensuring fast and trustworthy data access. We implement it as a lightweight and portable container-based monitoring system and evaluate it through empirical experiments. The results show that DEMon efficiently decimates and retrieves the monitoring information, addressing the abovementioned challenges.**

***Index Terms*—Edge computing, Monitoring Systems, IoT and Decentralized Storage and Retrieval, Trustworthy Systems.**

I. INTRODUCTION

Edge computing offers computing resources for the latency-sensitive Internet of Things (IoT) workloads enabling data processing at the network edge. However, unlike Cloud computing, which provides reliable and robust computing resources from centralized data centers, Edge computing offers services from a highly distributed environment with heterogeneous and resource-constrained compute and network resources [1], [2]. Therefore, deploying application services reliably on Edge needs efficient infrastructure monitoring systems, allowing applications and service providers to make crucial decisions based on the monitoring data.

Monitoring services allow observation of the overall status of the infrastructure and play a crucial role in resource management tasks such as resource provisioning, scheduling, load balancing, and failure detection. Traditionally, Cloud computing services are offered through data centers directly managed by single service providers, and data center resources are monitored through sophisticated Data Center Infrastructure Management (DCIM) tools centrally deployed on robust and reliable servers. Many of the Cloud service providers build their in-house DCIM platforms (e.g., Google's Borgmon), while private Clouds adopt open-source monitoring tools such as Zabbix [3] and Prometheus [4]. These systems query the state of the other servers and resources at predefined time intervals and store the monitored data in time-series databases, requiring centralized and reliable computational and storage resources, which are infeasible for the Edge due to its unique requirements and challenges.

First, Edge computing infrastructure is highly heterogeneous, consisting of IoT devices, embedded systems, domain-specific accelerators, and inexpensive off-the-shelf commodity servers and micro-data centers. Such hyper-heterogeneity with failure-prone devices introduces a massive complexity to the design and implementation of monitoring systems. Secondly, unlike a Cloud, which has a high-speed and reliable network, Edge infrastructures are built upon limited bandwidth and unreliable networks, including wireless and cellular networks. Thus, communication failures should be considered a norm rather than an exception. Thirdly, Edge infrastructures are volatile- where resources are pooled by multiple services providers across various network domains [2], and machines are dynamically provisioned or de-provisioned (join and leave the resource pool) based on network connectivity and power budget, among other parameters. Thus, creating challenges for retrieving trustable monitoring data in multi-party resource environments [5]. Some recent works have explored solutions for Edge monitoring [6]–[8] and multi-tier Fog computing [9], [10]. However, they consider either some form of a centralized controller or remote storage mechanisms.

In this paper, addressing the challenges associated with Edge monitoring, we propose Decentralized Edge Monitoring (DEMon), an efficient, decentralized, self-adaptive, and trustable monitoring system for a highly volatile Edge environment. We imagine the proposed system as a distributed information management system with efficient information spreading, storage, and data retrieval. We use a stochastic group communication protocol for information dissemination in a volatile Edge environment [11]–[13]. In particular, a Gossip-based information dissemination algorithm adjusting to the requirements of Edge environments. Our proposed approach effectively decimates information across the network without introducing massive concentrated network traffic and achieves uniform network load distribution. In addition, we propose the *Leaderless Quorum Consensus (LHC)* protocol for information retrieval, which can quickly aggregate the information of a specific node, ensuring fast and trustworthy retrieval of the data. The DEMon's architecture is decentralized, i.e., it does

978-1-6654-6087-3/22/$31.00 ©2022 IEEE
DOI 10.1109/UCC56403.2022.00026

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 20,2023 at 13:58:15 UTC from IEEE Xplore. Restrictions apply.

not depend on a centralized controller or servers for information storage and retrieval; instead, it uniformly distributes the data across the network autonomously. Furthermore, it is self-adaptive, i.e., no external configurations or measures are enforced during resource or network failures and infrastructure changes. These features of DEMon provide an efficient mechanism for decentralized information dissemination and storage in Edge. Moreover, unlike centralized monitoring services, application services can access the monitored data quickly without introducing increased latency.

II. Background and System Architectural Model

A. Gossip Protocol

The epistemic algorithms are inherently stochastic [12], [13] and provide a robust framework for building communication protocols and information systems in networked systems [11], [14], [15]. Gossip protocol is a popular epistemic-based algorithm that provides efficient means for group communication without broadcasting. It is highly scalable and resilient and avoids a single point of failure [12]. In this protocol, each node periodically selects a few other random nodes, exchanges the state information, and waits to receive data from other nodes. The rate of message exchange (`gossip_rate`) and the number of random nodes chosen (`gossip_count`) are configurable. Once a node receives state information, it updates its state if that particular data is new or not present. Otherwise, it drops the message and continues to wait for new messages. The system is considered converged if all the nodes know about every other node. However, in our case, since the monitored data also changes continuously in all nodes, gossiping should continue indefinitely. It has been shown that the Gossip protocol works well in designing the theoretical distributed systems [12]. However, its application in large-scale real-world systems is less explored [16]. In addition, if the hyper-parameters of the protocol (e.g., `gossip_rate` and `gossip_count`) are misconfigured, it might exponentially increase the network and storage load and may even perform worse than broadcast-based communication [17]. Therefore, it becomes imperative to carefully study the feasibility and analyze protocol in resource-constrained Edge. This paper investigates the effect of various hyper-parameters on metrics such as network load, query latency, and information quality.

B. Architecture of DEMon

Figure 1 shows a high-level architectural view of the DEMon with its main components where each Edge node is a physical or a virtual machine. The DEMon's architecture is categorized into two parts, i.e., *information dissemination* and *information retrieval*. In the first component, the sender process accesses local monitoring data from standard OS-specific interfaces. It then periodically sends its current system state, including its own state (monitoring metrics) and the state of other nodes' that it currently has. Similarly, the receiver process asynchronously waits for the new information sent from other nodes. The communication between sender and receiver is inter-node communication over the network, and

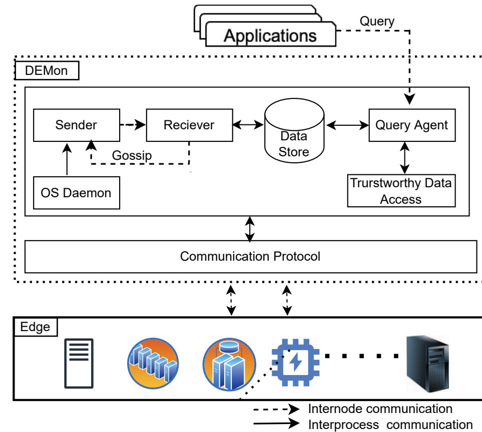


Fig. 1: DEMon Architecture

each node has both sender and receiver processes. The second component provides monitoring information for user requests. The query agent scans the local data store, and the trustworthy data verification mechanism is employed to provide the requested information. The same instance of DEMon runs in all the Edge nodes, and no node has any different role in the system. In the next section, we describe the essential components of DEMon in detail.

III. DEMon: Decentralised Edge Monitoring

A. Data Format

The primary data in our monitoring systems is the resource usage levels of Edge nodes. Periodically, the monitoring system collects essential utilization metrics such as CPU, memory, network, and storage capacity, with different granularity [3]. In addition, the DEMon collects required metadata (hostname), heartbeat (timestamp), and a time counter (integer). We also maintain a message digest of the whole monitored data, which is later used to establish the trustworthiness of monitoring data during the information retrieval. Each node's data is stored using a JSON object format, where *key* is the node's IP address, and *value* is collected metrics and associated metadata. A new node's information is easily added by creating a new *key* with its corresponding IP. A sample JSON object of a node is shown below.

```
{"nodeIP":
  {"appSate":{"cpu":"", "memory":"",
              "network":"", "storage":""},
   "counter":"",
   "hbState":""
   }
 "digest":""
}
```

B. Information dissemination

The Algorithm 1 describes the pseudo-code for our information dissemination logic. The Algorithm 1 has two

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 20,2023 at 13:58:15 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: Information dissemination using Gossip

```
1  X ← NULL
2  Communicator Thread
3  for every t seconds do
4      X ← updateOwnInfo(self.getCurrentState())
5      nodes ← selectRandomTargets()
6      for each node in nodes do
7          send X.metaData to node
8          if response from node then
9              updates, requests ← parse(response)
10             for each update in updates do
11                 X.update ← updateInfo(update)
12             end
13             send getKnownInfo(X.requests) to node
14     end
15 end
16 Listener Thread
17 while True do
18     receive Y.metadata from sender
19     X ← getKnownInfo()
       /* Compare the time counters*/
20     if X.metadata ≠ Y.metadata then
21         updates, requests ← parse
             (X.metdata, Y.metdata)
22         send updates, requests to sender
23         receive Y from sender
24         update X.Y
25 end
```

Algorithm 2: Trustworthy data retrieval based on Leaderless Quorum Consensus (LQC) Protocol

```
1  queryNodes ← selectQueryNodes()
2  responses ← queryMetaData(queryNodes)
3  R ← select quorum number responses
   /*provides a notion of  consistency*/
4  if compare (R.timestamp) is true then
       /*ensures data trustworthiness*/
5      if compare (R.degset) is true then
6          queryData()
7      else
8          go to selectQueryNodes()
9      end
10 else
11     go to selectQueryNodes()
12 end
```

main execution threads, sender, and listener. For each predefined time interval t (also known as gossip_rate), a sender thread reads its status from the local OS interface and updates its status in the object store (line 4). Then, it randomly chooses the gossip_count number of nodes from its membership list and initially only sends metadata (lines 5-7), which includes all the known node ids and their time counters. This logic avoids excessive bandwidth consumption by avoiding duplicate data transmission repetitively since the receiver only intends to receive missing or fresh data. The sender would then respond with the requested information and update its local data store if any new updates are sent from the receiver (lines 9-13).

Similarly, in the receiver thread, for any incoming message, it checks if its data store needs to be updated based on the metadata it has received, i.e., key (node ids) and time counter values (lines 18-20). Based on the comparison, the receiver sends back a response message. This response message includes two parts: first, requesting the new piece of missing information (requests). Second, if a receiver has new data in contrast to the sender, it directly sends this updated information to the sender (updates). This dual role of the receiver enables quick information sharing between a sender and a receiver where a sender is not only disseminating new information across the network but also simultaneously getting updated with the new information from the receiver. The instance of the same algorithm runs in all the Edge nodes. The stochastic selection of nodes in Algorithm 1 ensures the uniform distribution of messages in the network, and it does not spike a specific network link with high bandwidth usage. Moreover, the information spreads exponentially across the nodes and evenly distributes the network load [17]. Any permanent or transient failures do not affect the monitoring infrastructure as new nodes can send and receive the monitored data from the network and quickly know about all other nodes.

C. Storage

We store the data in an in-memory key-value object-store where each node maintains a hashmap with all nodes IPs as its *key*, and its corresponding monitored data as *value*, based on the format described in Section III-A. For an incoming gossip message, we check if it has any new data for a node, comparing the current time counter to the received message and update its local data store. Thus, the hashmap size remains constant at any time unless new nodes are added to the system, allowing greater scalability for our monitoring system. In addition, the key-value object store provides interoperability across different programming interfaces and flexibility to build various aggregate and query functions. One can also checkpoint the data at regular intervals for persistent data storage; however, we do not checkpoint the data.

D. Information Retrieval

Since the data is replicated across the participant nodes, it is crucial to ensure that retrieved data is recent, consistent, trustworthy, and not altered by malicious nodes. Generally, consistency is ensured in distributed information systems by using consensus protocols like Paxos [18]. However, such protocol requires leader nodes and is computationally expensive, making them infeasible in our case. Consequently, we propose an efficient group-based *Leaderless Quorum Consensus (LQC)* protocol as shown in Algorithm 2, where we eliminate centralized leader node architecture, which creates

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 20,2023 at 13:58:15 UTC from IEEE Xplore. Restrictions apply.

failure bottleneck and increases latency. Here, we parallelly query a subset of nodes for required information, and the responses consist of the queried data along with the associated message digests (lines 1-3), and we wait until we receive the minimum number of responses, i.e., quorum. The data is considered consistent and trustworthy when the time counters and the corresponding digests (ensuring data integrity) are matched from the responses (lines 4-5). Otherwise, the current request session will be discarded, and a new set of nodes will be chosen randomly (lines 7-11). This lightweight protocol ensures a notion of weak consistency with a significant performance advantage and provides the required reliability for a volatile Edge environment. Such leaderless protocols have found applications in many recent distributed database systems, such as Cassandra, allowing faster transactions.

IV. Performance Evaluation

The proposed monitoring framework is implemented as a multithreaded application with Python language, realizing Algorithm 1 and Algorithm 2. The monitoring process creates a main thread controlling the application's business logic, while two other threads, sender and receiver, are used for gossiping, i.e., for message sending and receiving. Each node collects its own utilization metrics in predefined intervals using the "`psutil`" tool, updates the system state, and increments its time counter. The entire application is containerized using docker and deployed as a background daemon in all the Edge nodes, and a lightweight FLASK framework is used for establishing the REST APIs communication among the Edge nodes. This makes our DEMon agent lightweight and interoperable for resource-constrained heterogeneous devices with a small resource footprint.

A. Experimental Setup

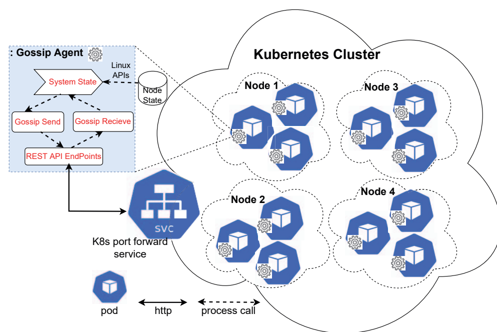


Fig. 2: DEMon implementation on Kubernetes emulating large scale Edge environment, each pod representing an edge node

We deployed our system on the Kubernetes (K8s) cluster and configured the system to emulate real-world Edge infrastructure, as shown in Figure 2, a schematic representation of our testbed. The K8's cluster has four nodes, each having a resource allocation of 8 cores and 8 GB of memory. Each pod in the Kubernetes cluster represents an Edge node, and our containerized application runs inside these pods. It is important to note that no changes are required to deploy the DEMon on the new Edge infrastructure. We scaled Edge nodes (i.e., pods) from 0 to 150. We utilize Kubernetes port forwarding services for the pods, enabling direct communication between all pods. We conducted experiments with different hyperparameter values, including `gossip_count` and `gossip_rate`, and analyzed the results. The following important metrics are measured.

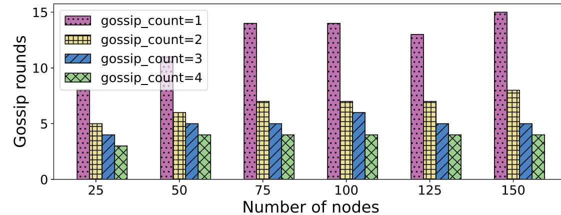


Fig. 3: Convergence vs rounds (gossip rate =3)

Convergence: This represents the time at which when every node knows about every-other node from the start of the monitoring process. In our case, gossiping continues forever to spread recent information across the system.

Query latency: This represents the number of messages required to retrieve the information about any particular node based on Algorithm 2.

Age of Information: This represents the timeliness (freshness) of the information stored in each node (as monitoring information is continuously updated). We measure this by using the Age of Information (`AoI`) metric [19], defined as:

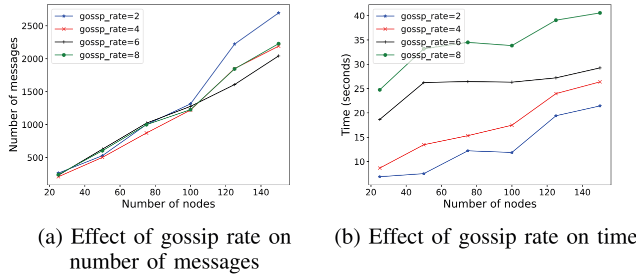
$$AoI = \frac{1}{n}\sum_{i=1}^{n} t_i - u(t_i) \quad (1)$$

where n is the total number of other node's information a current node has, and t_i is the actual time counter of remote $node_i$, and $u(t_i)$ is the current time counter of $node_i$. To illustrate, let us assume two nodes, `node_1` and `node_2`. The `AoI` of `node_1` is the difference between the time counter of `node_2` at `node_1`, i.e., $u(t)$ and the actual time counter at `node_2` itself, i.e., t.

B. Results and Analysis

1) Convergence Analysis: In this experiment, we scaled the number of Edge nodes (pods) upto 150 with an interval of 25. All experiments are repeated three times, and average values are reported. Figure 3 shows the number of gossip rounds required to reach the initial convergence state (when each node knows every other node). Since each node has its gossip round running locally, we consider the maximum gossip round among all nodes as system convergence. Here, we configured `gossip_count` $= \{1, 2, 3, 4\}$ and `gossip_rate` to 3. The higher value of `gossip_count` spreads the information quickly, and converges faster, which denotes that when a new node (re)joins, it is able to know about all other nodes in fewer

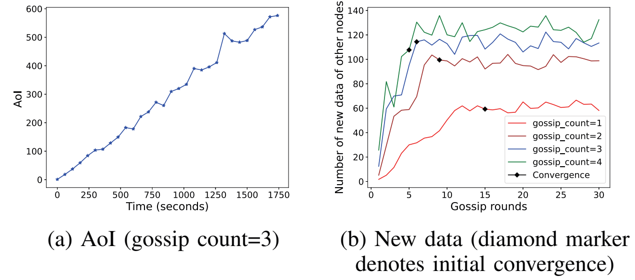
Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 20,2023 at 13:58:15 UTC from IEEE Xplore. Restrictions apply.



(a) Effect of gossip rate on number of messages

(b) Effect of gossip rate on time

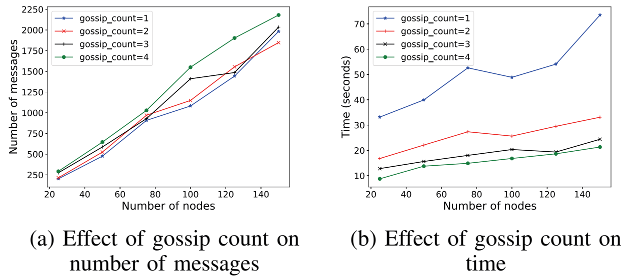
Fig. 4: Sensitivity analysis of gossip rate parameter



(a) AoI (gossip count=3)

(b) New data (diamond marker denotes initial convergence)

Fig. 6: Analysis of AoI and new data updates (system size=150, gossip rate=3)



(a) Effect of gossip count on number of messages

(b) Effect of gossip count on time

Fig. 5: Sensitivity analysis of gossip count parameter

gossip rounds. For instance, when gossip `gossip_count=4`, the system converges within four rounds.

Figure 4 depicts the effects of `gossip_rate` on the number of messages and time during the initial system convergence. Here, each node's monitoring interval and `gossip_rate` are set to similar; thus, for each `gossip_rate` second, a node sends its new state. We configured `gossip_rate` $= \{2, 4, 6, 8\}$ seconds and `gossip_count` to 3. When the number of nodes increases, the total number of messages simultaneously increases, demonstrating that a higher number of message exchanges are required for a large number of nodes, as shown in Figure 4a. Irrespective of `gossip_rate`, the number of messages is similar across all node sizes; representing convergence requires an almost equal number of messages for fixed system size and does not depend on `gossip_rate`. However, `gossip_rate` affects the number of messages per second. Therefore, if we want to control the bandwidth usage or speed of convergence, `gossip_rate` can be configured accordingly. Figure 4b shows the effect of `gossip_rate` on convergence time. The smaller `gossip_rate` results in faster convergence time and vice versa.

Similarly, `gossip_count` (i.e., decides number of random nodes chosen for each gossip), also significantly affects the performance, as seen in Figure 5. Here, we configured `gossip_count` $= \{1, 2, 3, 4\}$ and `gossip_rate` to 3. The number of messages highly depend on system size (see Figure 5a). Consequently, `gossip_count` value strongly affects the convergence time, as seen in Figure 5b.

Therefore, if faster convergence is required, `gossip_count` should be set to a higher value and `gossip_rate` to a smaller value. If minimal bandwidth consumption is necessary, contrast values can be set to these parameters. Hence, allowing greater flexibility to configure according to the Edge environment needs.

2) Information Retrieval: The querying process follows the logic explained in the Algorithm 2, where `quorum_number` is set to 3. We generate `quorum_number` of parallel queries, requesting the utilization metrics of a random node. Once a node receives a request, it checks the local object store based on the requested node IP as key and sends the data in the format as described in Section III-A. We set different failure rates from 0%-90% with the interval of 10% by randomly disconnecting the `failure_rate` % of nodes for each setting. We performed 100 queries each time for different failure rates. The DEMon can remarkably provide the information of a requested node without query failures. This is because, since every node stores information of every other node, even when 90% of the nodes fail, the queried node information is successfully retrieved. We received the queried information with a maximum number of messages of 19 and a minimum number of messages of 3 (best case scenario, equal to `quorum_number`), with an average value of 4.65.

3) Age of Information (AoI) Analysis: In this experiment, we take a snapshot of the monitoring data from all nodes every minute for 30 minutes. Once the system is converged initially, for each interval, each node's `AoI` is calculated based on Equation 1. Figure 6a shows the overall `AoI` of the system (average from all the nodes). The `AoI` gradually increases as the time interval increases. This behavior is expected since every node gossips to a very few nodes for each `gossip_rate` seconds; it would take several further gossip rounds to reach the updated information to other nodes, while each node simultaneously updates their monitoring information, creating a time delay between the two data instances. However, as Figure 6b shows, the monitoring agents receive the newly updated information of other nodes in each round, and the rate of new updates is highly dependent on the `gossip_count`. When `gossip_count` is set to 4, the average number of new fresh data updates is almost equal to the system size (i.e., 150). Therefore, if the most recent or fresh data is needed, higher `gossip_count` values can be configured.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 20,2023 at 13:58:15 UTC from IEEE Xplore. Restrictions apply.

V. Related Work

Traditional monitoring systems in Cloud data centers such as Google's Borgmon [20], and Prometheus [4] are mainly centralized systems. On the other hand, existing Edge computing platforms such as Kubernetes and KubeEdge depend on a centralized control plane for monitoring, making them infeasible for critical Edge infrastructures requiring strict latency and reliability under failures. Researchers have made many efforts to address the challenges in Edge monitoring. In [6], authors proposed "PyMon" to monitor container-based computing architectures with a small resource footprint. The solution is primarily targeted at IoT-based single-board Edge devices. Similarly, the authors in [7] propose a network monitoring approach for data streaming applications, where each Edge node hosts a monitoring probe and pushes the data to a centralized time-series database. The FMonE [8] proposes the design of monitoring solutions considering elasticity and resiliency and solving the unique challenges of Edge systems. However, its storage and information processing depends on centralized database systems. Some works have also explored self-adaptive monitoring for multi-tier Fog computing systems [9], [10]. The FogMon [10] and AdaptiveMon [9] proposes hierarchical P2P architecture where lower-tier nodes are dedicated as followers and higher-tier nodes as leaders. All these solutions exhibit a partially centralized architecture in the form of leader nodes or aggregation functions.

Gossip protocol in monitoring: Gossip protocols have found use cases in developing monitoring systems. Ward et al. [21] propose monitoring large-scale cloud systems with layered Gossip protocols grouping the resources into multiple layers, from VMs to multi-region data centers. Similarly, *Astrolabe* [14] is one of the earlier systems implemented based on the Gossip protocol for distributed information management. With the combination of peer-to-peer Gossip protocol, mobile code, and SQL query language, the system is implemented to manage the data collection, storage, and aggregation in real-time by organizing resources in hierarchical domains. In addition, the authors in [22] use the Gossip protocol to monitor network-wide aggregates (such as AVERAGE, MIN, MAX). However, the main focus in these works is on aggregating the information of resources, and they do not address a complete monitoring system's requirements. Consequently, our objective is to provide a framework and techniques for self-adaptive, self-configurable, and trustworthy monitoring for volatile Edge environments without limiting to a specific application task.

VI. Conclusions

The existing monitoring systems are centralized in architecture, which increases information storage and retrieval latency and creates failure bottlenecks, making them infeasible for volatile Edge environments. To that end, we present a framework called DEMon that is designed to work autonomously without any external configurations and stores the monitoring information in a decentralized manner. DEMon achieves this through efficient information spreading based on the Gossip protocol and configuring hyperparameters based on the system properties. In addition, DEMon provides a trustable information retrieval mechanism. A lightweight and interoperable container-based prototype system is implemented. The experimental results on the Kubernetes cluster show that DEMon quickly spreads the monitored information and can retrieve the information even when most of the nodes fail.

Acknowledgment

This work is partially funded through the RUCON project (Runtime Control in Multi Clouds), Austrian Science Fund (FWF): Y904-N31 START-Programm 2015 and the SWAIN project, CHIST-ERA grant: CHIST-ERA-19-CES-005, FWF: I 5201-N.

References

[1] M. Satyanarayanan, W. Gao, and B. Lucia, "The computing landscape of the 21st century," in *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, 2019.

[2] R. Buyya, S. N. Srirama *et al.*, "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM computing surveys (CSUR)*, vol. 51, no. 5, pp. 1–38, 2018.

[3] R. Olups, *Zabbix Network Monitoring.* Packt Publishing Ltd, 2016.

[4] J. Turnbull, *Monitoring with Prometheus*. Turnbull Press, 2018.

[5] S. Forti and et al., "Secure cloud-edge deployments, with trust," *Future Generation Computer Systems*, vol. 102, pp. 775–788, 2020.

[6] M. Großmann and C. Klug, "Monitoring container services at the network edge," in *29th International Teletraffic Congress*, 2017.

[7] S. Taherizadeh, I. Taylor *et al.*, "A network edge monitoring approach for real-time data streaming applications," in *Economics of Grids, Clouds, Systems, and Services*. Springer, 2017.

[8] Á. Brandón, M. S. Pérez *et al.*, "Fmone: A flexible monitoring solution at the edge," *Wireless Communications and Mobile Computing*, 2018.

[9] V. Colombo, A. Tundo *et al.*, "Towards self-adaptive peer-to-peer monitoring for fog environments," in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2022.

[10] S. Forti, M. Gaglianese, and A. Brogi, "Lightweight self-organising distributed monitoring of fog infrastructures," *Future Generation Computer Systems*, vol. 114, pp. 605–618, 2021.

[11] A. Demers, D. Greene *et al.*, "Epidemic algorithms for replicated database maintenance," in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, 1987, pp. 1–12.

[12] K. Birman, "The promise, and limitations, of gossip protocols," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 5, pp. 8–13, 2007.

[13] H. van Ditmarsch, J. van Eijck *et al.*, "Epistemic protocols for dynamic gossip," *Journal of Applied Logic*, vol. 20, pp. 1–31, 2017.

[14] R. Van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM transactions on computer systems*, vol. 21, 2003.

[15] R. Azimi and H. Sajedi, "A decentralized gossip based approach for data clustering in peer-to-peer networks," *Journal of Parallel and Distributed Computing*, vol. 119, pp. 64–80, 2018.

[16] H. Ditmarsch, D. Grossi *et al.*, "Parameters for epistemic gossip problems," in *LOFT 2016-12th Conference on Logic and the Foundations of Game and Decision Theory*, 2016.

[17] K. R. Apt, E. Kopczynski, and D. Wojtczak, "On the computational complexity of gossip protocols," in *IJCAI*, 2017.

[18] L. Lamport, "Paxos made simple," *ACM SIGACT News (Distributed Computing Column) 32, 4*, pp. 51–58, 2001.

[19] A. Arafa, R. D. Yates, and H. V. Poor, "Timely cloud computing: Preemption and waiting," in *57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2019.

[20] B. Beyer, C. Jones *et al.*, *Site reliability engineering: How Google runs production systems.* " O'Reilly Media, Inc.", 2016.

[21] J. S. Ward and A. Barker, "Monitoring large-scale cloud systems with layered gossip protocols," *arXiv preprint arXiv:1305.7403*, 2013.

[22] F. Wuhib, M. Dam *et al.*, "Robust monitoring of network-wide aggregates through gossiping," in *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007, pp. 226–235.

Authorized licensed use limited to: TU Wien Bibliothek. Downloaded on October 20,2023 at 13:58:15 UTC from IEEE Xplore. Restrictions apply.