AdapPF: Self-Adaptive Scrape Interval for Monitoring in Geo-Distributed Cluster Federations

Chih-Kai Huang, Guillaume Pierre

AdapPF: Self-Adaptive Scrape Interval for Monitoring in Geo-Distributed Cluster Federations

Chih-Kai Huang
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
chih-kai.huang@irisa.fr

Guillaume Pierre
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
guillaume.pierre@irisa.fr

***Abstract*—Monitoring plays a vital role in geo-distributed cluster federation environments to accurately schedule applications across geographically dispersed computing resources. However, using a fixed frequency for collecting monitoring data from clusters may waste network bandwidth and is not necessary for ensuring accurate scheduling. In this paper, we propose Adaptive Prometheus Federation (AdapPF), an extension of the widely-used open-source monitoring tool, Prometheus, and its feature, Prometheus Federation. AdapPF aims to dynamically adjust the collection frequency of monitoring data for each cluster in geo-distributed cluster federations. Based on actual deployment in the geo-distributed Grid'5000 testbed, our evaluations demonstrate that AdapPF can achieve comparable results to Prometheus Federation with 5-seconds scrape interval while reducing cross-cluster network traffic by 36%.**

***Index Terms*—Monitoring, Geo-distributed cluster federations, Prometheus, Self-adaptive, Fog computing.**

I. Introduction

The emergence of distributed computing paradigms such as fog and edge computing presents new opportunities to serve end users in closer proximity and process data from sources outside traditional on-premise data centers. To improve the operational efficiency of this new geo-distributed context, cloud-native technologies such as containerized applications and Kubernetes are becoming increasingly popular [1]. However, as the system size grows, it is crucial to consider the potential impact on system performance. For instance, Kubernetes API server can stall [2] when the system size is large enough, making it challenging to scale a single Kubernetes cluster to a higher degree.

To address issues related to performance and scalability, deploying multiple clusters has emerged as a viable solution. This concept aims to deploy a large number of small clusters in different strategic locations to provide services and process nearby data. To enable administrators to treat these independent geo-distributed clusters as a single homogeneous cluster, it is beneficial to group these clusters into a *federation* [3]. A federation platform is typically organized with one management cluster and several member clusters. The management cluster determines which member clusters will handle each newly deployed application. To enable accurate scheduling decisions, it is necessary to have information on the resource usage status of each member cluster in the federation [4]. This requires a robust monitoring framework that can provide resource utilization data, such as Prometheus [5] and its extension Prometheus Federation [6]. When deployed in a cluster federation, Prometheus Federation periodically *scrapes* (i.e., retrieves) the relevant metrics values from each member cluster.

We demonstrate in this paper that the frequency of collecting monitoring data affects the accuracy of scheduling applications across geo-distributed cluster federations. If the member clusters are mostly idle, the scheduler can easily select among any of them because their computing resources will have enough capacity to execute applications. This case does not require real-time monitoring data of these member clusters, which gives opportunities for low cross-cluster network traffic. However, when computing resources in member clusters are in high demand, minimizing the number of pending pods[1] in member clusters requires one to carefully determine which clusters should run the applications. This situation requires timely monitoring of data from these member clusters, whose collection requires more network traffic. Additionally, we note that the scrape interval[2] in Prometheus Federation is a *fixed* value, which could potentially waste network bandwidth and cannot support accurate scheduling. Therefore, establishing an appropriate scraping interval is critical to balancing the imperative of gathering timely monitoring data for accurate scheduling while reducing cross-cluster network traffic resulting from metrics scrapes.

This paper proposes Adaptive Prometheus Federation (AdapPF), an extension of Prometheus Federation which dynamically adjusts the scrape interval of the target member cluster based on its resource status to balance between cross-cluster network traffic and the required accuracy of monitoring data. To this end, a self-adaptive scrape interval method tailored for AdapPF considers the status of CPU and memory computing resources. When the targeted member cluster utilizes a substantial amount of resources, the scrape interval will automatically adjust, increasing the frequency of data collection. This allows for timely acquisition of up-to-date monitoring data, enabling the scheduler or system alarm to

[1]In the event that there are insufficient resources on the nodes within the member cluster to run the pod, the pod will be placed in a pending state until adequate resources become available.

[2]Scrape interval refers to the time duration between two consecutive data pulls from the designated targets or endpoints.

make informed decisions or trigger alerts earlier. We show, using actual deployments, that AdapPF achieves comparable scheduling accuracy to Prometheus Federation while reducing cross-cluster network traffic by up to 36%.

The remainder of this paper is structured as follows: Section II discusses the motivation behind this work, while Section III provides the background and related work. In Section IV, we describe the design of AdapPF and introduce the self-adaptive scrape interval approach. We evaluate our strategies in Section V, and summarize the paper's conclusions in Section VI.

II. Motivation

Accurate application scheduling in a geo-distributed cluster federation environment requires monitoring information from all member clusters to be collected and transferred to the management cluster. Prometheus, a well-known open-source monitoring tool, is stable enough for deployment in production environments and also suits to integrate with the federated scheduler for geo-distributed cluster federation environments [4]. However, the *fixed* scrape interval in both Prometheus and Prometheus Federation can lead to resource wastage. When the system load in each member cluster is relatively low, frequent scraping for timely data collection is unnecessary for scheduling and may result in excessive network bandwidth usage.

To illustrate this issue, we leverage an actual deployment in the geo-distributed Grid'5000 testbed [7]. In the setup, we launch 6 Kubernetes clusters, one of which is our management cluster (global view cluster), while the other five are member clusters. Each member cluster has five worker nodes with 2 CPU cores and 8 GiB of memory. We install mck8s [4] on the management cluster to manage member clusters. mck8s is an orchestrator for geo-distributed multi-cluster environments which uses Prometheus Federation as its monitoring solution. Furthermore, mck8s provides advanced scheduling policies based on the resource status of member clusters. By using mck8s, we can understand how different scrape intervals for Prometheus Federation may affect the scheduling accuracy when we inject workloads. We inject two workload datasets based on Google cluster-usage traces [8]. One dataset represents high resource usage for the platform and is identical to the Google cluster-usage traces. We skip the deployments with even indexes for the other dataset, which therefore represents low usage. We inject each workload for 60 minutes and wait 30 minutes to release the computing resources, resulting in 1,095 tasks for the high resource usage scenario and 547 tasks for the low usage scenario. We run each experiment ten times and calculate the percentage of pending pods with two different scrape interval for Prometheus Federation: 5 and 60 seconds.

Figure 1 shows the results of one of these 10 experiment rounds (number 10 in the Table) when we inject high workload, and Table I presents the average percentage of pending pods over time for all 10 experiment rounds (sorted). We observe that the percentage of pending pods is much lower when we set the scrape interval to 5 seconds compared to

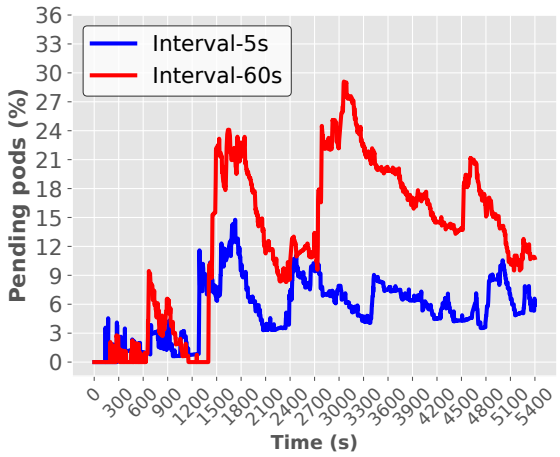


Fig. 1. Percentage of pending pods (high workload injection)

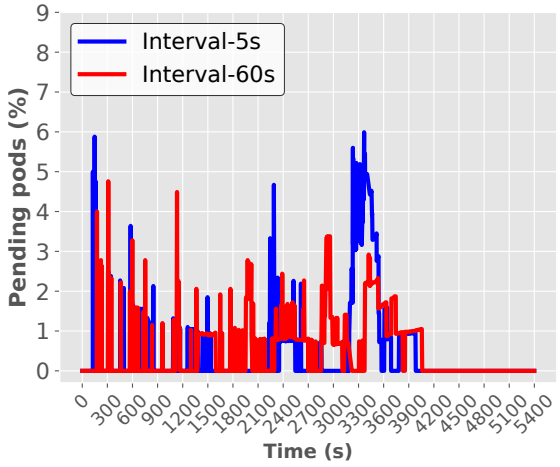


Fig. 2. Percentage of pending pods (low workload injection)

60 seconds. However, the cross-cluster network traffic of these two cases is 55.32 and 4.63 KiB/sec, respectively. The difference between these two cases is one order of magnitude. When the platform size grows, network traffic for monitoring will grow proportionally and may ultimately represent the majority of the system management traffic [9]. We also note that when we reduce the workload, the percentage of pending pods is similar regardless of whether the scrape interval is set to 5 or 60 seconds, as illustrated in Figure 2. These results indicate that in this case a 60-seconds scrape interval can achieve similar results without precise information about the member clusters since clusters have enough resources to handle the workload.

This scheduling experiment inspires this work: we aim to find a good balance between cross-cluster network traffic and accurate monitoring data to enable accurate application scheduling by dynamically adjusting the scrape interval based on the resource status of member clusters. Doing so can achieve precise scheduling decisions while using lower cross-cluster network traffic than the fixed interval strategy of Prometheus Federation.

TABLE I
10 RESULTS OF PENDING PODS PERCENTAGE (AVG: AVERAGE OF 10 ROUNDS; STD: STANDARD DEVIATION OF 10 ROUNDS). LOWER VALUE IS BETTER

	1	2	3	4	5	6	7	8	9	10	Avg	Std
High Workload												
interval-5s (%)	1.88	2.98	3.50	3.88	4.13	4.45	4.54	4.91	5.00	5.22	4.05	1.03
interval-60s (%)	4.66	4.90	6.05	6.45	6.78	6.84	6.84	7.26	11.62	12.13	7.35	2.53
Low Workload												
interval-5s (%)	0.29	0.31	0.32	0.41	0.42	0.86	0.87	0.87	0.88	1.57	0.68	0.41
interval-60s (%)	0.39	0.39	0.46	0.58	0.81	0.90	0.99	1.01	1.12	1.30	0.79	0.32

III. BACKGROUND AND RELATED WORK

Fog computing technology deploys computing resources in the network edge, close to the end users and the location of data source generated by Internet of Things (IoT) devices [10]. This design enables computational tasks to perform close to the data source, reducing the volume of data transmitted from IoT devices to data centers for analysis. Some previous studies based on a single disturbed cluster as their solution and focus on different aspects of fog or edge computing such as placement [11], service caching [12], [13], and migration [14]. However, fog computing platforms have geographically distributed computational resources and many fog nodes. A single disturbed cluster may encounter scalability issues when the size of a fog computing platform covers a whole country or even a continent. To confront this scalability issue, other related works consider multiple geo-distributed cluster environments and address particular topics such as task scheduling [15], resource management [4], [16] and fault prediction [17]. All these studies leverage monitoring tools to reach their goals. However, they did not specifically focus on the issue of monitoring in a geo-distributed cluster federation environment.

Monitoring is an essential function in modern cloud data centers. Many open-source or commercial monitoring solutions perform well in the traditional cloud environment, including DARGOS [18], JCatascopia [19], Nagios [20], PCMONS [21], and Zabbix [22]. However, these works are designed for something other than the fog computing platform since they did not consider the specific characteristics of fog computing [23]. To overcome the challenges of monitoring in a fog computing environment, there are some solutions such as FMonE [24], FogMon [25], PyMon [26], and Prometheus [5]. Among them, Prometheus is a "graduated project" as announced by the Cloud Native Computing Foundation (CNCF). The graduated project maturity level shows that Prometheus is stable for production and has great potential to integrate with modern orchestrators such as Kubernetes. Prometheus has also become the monitoring solution for many research works [27]–[32].

Prometheus Federation is a distributed monitoring function offered by Prometheus. This feature allows a Prometheus server to gather monitoring data from other servers, which can thus build a global-view cluster and scale up to monitor a large-scale distributed platform. By querying the monitoring metrics from the global-view cluster, the administrators can easily monitor the status of other clusters instead of accessing each cluster individually. Prometheus Federation has been successfully used for monitoring systems deployed across geographically distributed environments [4], [33]–[36].

However, Prometheus Federation may produce high cross-cluster network traffic across the federated clusters when the system size is large, and the frequency of collecting monitoring data is high. Additionally, these federated clusters often deploy to build geo-distributed platforms that cover a whole country or even a continent. The higher network traffic across such a large area may lead to a waste of network bandwidth. We also note that Prometheus Federation scrapes the metrics from the target clusters using a *fixed* scrape interval. Increasing the frequency of metric scraping by using a shorter scrape interval can enhance the accuracy of monitoring data of target clusters in the global-view cluster. However, this comes with the downside of generating more cross-cluster network traffic.

The closest proposed to our work is AdaptiveMon [37], which presents self-adaptive monitoring for fog environments. AdaptiveMon can adjust the frequency of reported metrics in a hierarchical peer-to-peer monitoring architecture. However, this solution was not designed for cluster federation environments. Moreover, AdaptiveMon, an extension of FogMon [25], still needs to be proven suitable for integration with modern orchestration [23].

To limit network overhead and optimize the scrape interval, we present AdapPF, which can dynamically adjust the values of the scrape interval in a geo-distributed cluster federation environment. AdapPF checks the current resource status of the target cluster and adjusts the scrape interval. This work aims to reduce cross-cluster network traffic while maintaining the accuracy of monitoring data.

IV. SYSTEM DESIGN

The goal of this work is to achieve precise monitoring data for accurate application scheduling with lower cross-cluster network traffic in geo-distributed cluster federations. This section presents the design of Adaptive Prometheus Federation (AdapPF) and introduces dynamically adjusting scrape interval strategies specifically designed for AdapPF to achieve our objectives.

A. System model and architecture

A fog computing platform is a decentralized paradigm with a large number of fog nodes that are typically weak and

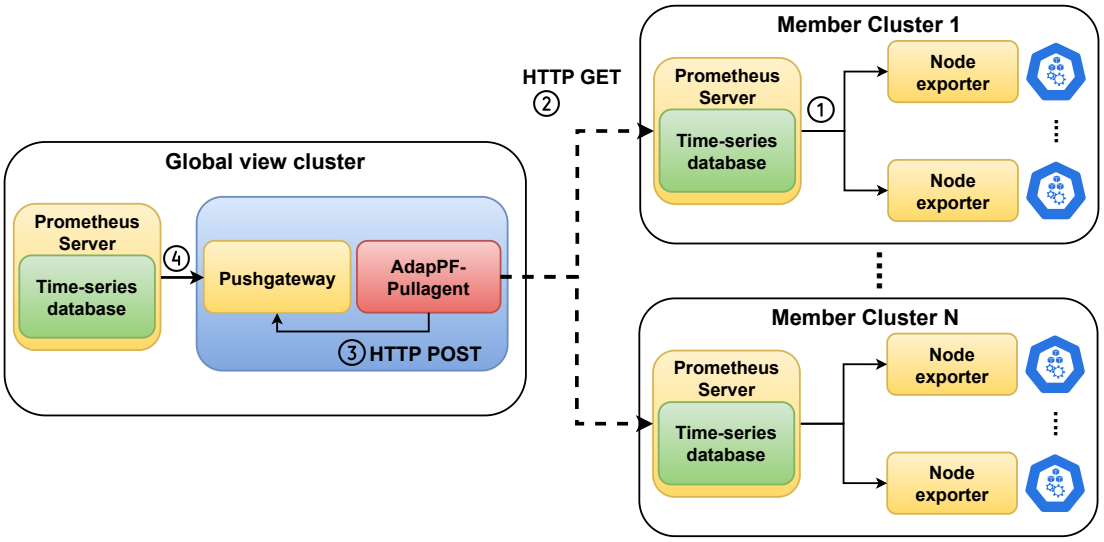


Fig. 3. Overview of AdapPF architecture and system workflow.

unstable. To address these characteristics, as discussed in Section III, we assume the existence of multiple clusters grouped into a cluster federation. The primary purpose of monitoring is to track the resource usage of computing nodes, and the information can be used for making scheduling decisions. The cluster federation distributes each cluster across various locations within a vast region and situates all servers in each cluster within the same area. Additionally, we assume that each server in a cluster has sufficient computing resources to run the required applications for monitoring. All clusters and servers can communicate with one another through the network. For this study, we select one cluster as our global-view cluster, while the remaining clusters are considered member clusters. Although the current design can support multiple tiers, in this paper we have chosen a two-tier architecture for simplicity.

We build AdapPF on the well-known Prometheus open-source monitoring solution and its feature Prometheus Federation as Prometheus is a mature monitoring solution for gathering resource usage data. We leverage the Prometheus server, Node-exporter [38], and Pushgateway [39] to build AdapPF. We present the overview of the architecture in Figure 3 and discuss the detailed description of each component below:

- Prometheus server: Prometheus servers are responsible for collecting monitoring data from the configured targets and storing these metrics in the local time-series database installed in all clusters. The administrator can retrieve and analyze monitoring data by querying the relevant Prometheus server. In the global-view cluster, the Prometheus server includes monitoring data from member clusters so that the users or federation's scheduler can have the resource status of all federated clusters to execute their jobs accordingly.
- Node-exporter: This application is a monitoring agent that exposes various metrics related to server resources, such as CPU and memory usage. Prometheus server can scrape metrics from Node-exporter to gather information about the status of individual machine resources. We install a Node-exporter on all nodes in all clusters.
- Pushgateway: Pushgateway provides an HTTP endpoint for the Prometheus server to scrape and cache the metrics. Pushgateway can also automatically generate alerting metrics for failed pushes. Administrators can set related alerting rules to receive notifications.

The AdapPF framework adds a specialized proxy in this architecture called AdapPF-Pullagent to facilitate the achievement of the self-adaptive scrape interval for each member cluster based on cluster resource status. Modifying the configuration files directly on the Prometheus server in the global-view cluster requires reloading it, which may cause system instability. Instead, the main purpose of AdapPF-Pullagent is to scrape metrics from the Prometheus server in each member cluster. A timer determines the timing for scraping metrics from target member clusters. Once AdapPF-Pullagent has scraped the metrics, it adds corresponding information to identify the member cluster to the data and pushes these metrics to Pushgateway. Administrators can launch multiple AdapPF-Pullagent applications to accommodate more member clusters, making the system more scalable. The detailed workflow is as follows:

Step 1: The Prometheus server in each member cluster periodically scrapes the metrics from Node-exporter and saves these monitoring data in the local database. This is a local operation within each member cluster.

Step 2: When the timer in AdapPF-Pullagent for target clusters expires, the Pullagent uses the HTTP GET method and Prometheus Federation API to scrape the metrics from the Prometheus server in the target member clusters. We use Gzip to compress the data transmission between the global-view cluster and member clusters.

Step 3: After AdapPF-Pullagent receives the metrics, the Pullagent adds related information (the IP address of the control plane set by the user) to the metrics to correlate them with the member cluster. Then, the Pullagent leverages the HTTP POST method to push these metrics to the Pushgateway.

Step 4: The Prometheus server in the global-view cluster periodically scrapes metrics from the Pushgateway at a user-defined interval and stores them in a local time-series database. As a result, this Prometheus server includes monitoring data coming from the target member clusters.

B. Self-adaptive scrape interval

To achieve a self-adaptive scrape interval for each member cluster based on the resource status in a geo-distributed cluster federation environment, AdapPF-Pullagent follows a classical Monitor, Analyze, Plan, Execute (MAPE) loop pattern. Each member cluster runs its own control loop for monitoring and self-adaptation. This design ensures that the AdapPF-Pullagent can adjust the scrape interval for each member cluster according to its local resource utilization status. We discuss each phase below.

1) Monitor: The Monitor phase gathers monitoring data in each member cluster, such as CPU or memory usage. As discussed in the previous section, The Pullagent scrapes the monitoring data when the timer in AdapPF-Pullagent for a target cluster counts down to 0. Therefore, AdapPF-Pullagent maintains the *current* monitoring data from the target member cluster. As a result, we can use these monitoring metrics for the next step.

2) Analyze: The main purpose of Analyze step is to process monitoring data that collect from the Monitor phase. In the current design, we leverage the following three metrics:

- *node cpu seconds total* is the cumulative amount of CPU time consumed by a node since its boot phase.
- *node memory MemFree bytes* is the amount of free memory on the node, measured in bytes.
- *node memory MemTotal bytes* is the total installed memory on the node, measured in bytes.

Prometheus Federation collects monitoring data from member clusters at the node level, providing metrics from each node. To analyze these three metrics, we calculate the average values of these three metrics independently across all servers for the target member cluster to obtain an overall representation of the cluster status and then compute the current resource usage as a percentage. Consequently, we obtain two values, CPU and memory utilization, and select the greater value to plan the scrape interval for the target member cluster. For instance, if the CPU usage is 60% and the memory usage is 48%, the system will select 60% as the Cluster Status (CS) for the subsequent step.

3) Plan: The Plan step uses the Cluster Status (CS) from the previous phase and calculates the scrape interval for the next round. We apply the following equation in each iteration:

$$M = \frac{Tmin_{cluster} - Tmax_{cluster}}{Rmax_{cluster} - Rmin_{cluster}} \tag{1}$$

$$intercept = Tmax_{cluster} - M \times Rmin_{cluster} \tag{2}$$

$$Interval_{cluster} = M \times CS + intercept \tag{3}$$

$Tmin_{cluster}$ and $Tmax_{cluster}$ represent the shortest and longest scrape interval for the target cluster, respectively. Similarly, $Rmax_{cluster}$ and $Rmin_{cluster}$ correspond to the highest and lowest resource status of the clusters that map to the scrape interval. For example, if the status of the current resource CS of the target cluster reaches $Rmax_{cluster}$, then the system sets the scrape interval to $Tmin_{cluster}$. This design uses shorter scrape intervals during high target cluster activity periods to use timely monitoring data for accurate application scheduling. At the same time, if the CS is $Rmin_{cluster}$, our method sets the scrape interval to $Tmax_{cluster}$ and saves the network bandwidth with a longer scrape interval.

In addition, the scrape interval must be between $Tmax_{cluster}$ and $Tmin_{cluster}$. Therefore, $Interval_{cluster}$ should satisfy the constraint:

$$Tmin_{cluster} \leq Interval_{cluster} \leq Tmax_{cluster} \tag{4}$$

Enforcing this constraint ensures that if the threshold is greater than $Rmax_{cluster}$ or less than $Rmin_{cluster}$, the scrape interval will remain for $Tmin_{cluster}$ and $Tmax_{cluster}$, respectively.

4) Execute: After calculating the scrape interval for the next round, the system stores this information in memory. When the timer for the target cluster expires again, our approach returns to the monitor step and finds the scrape interval for the next round.

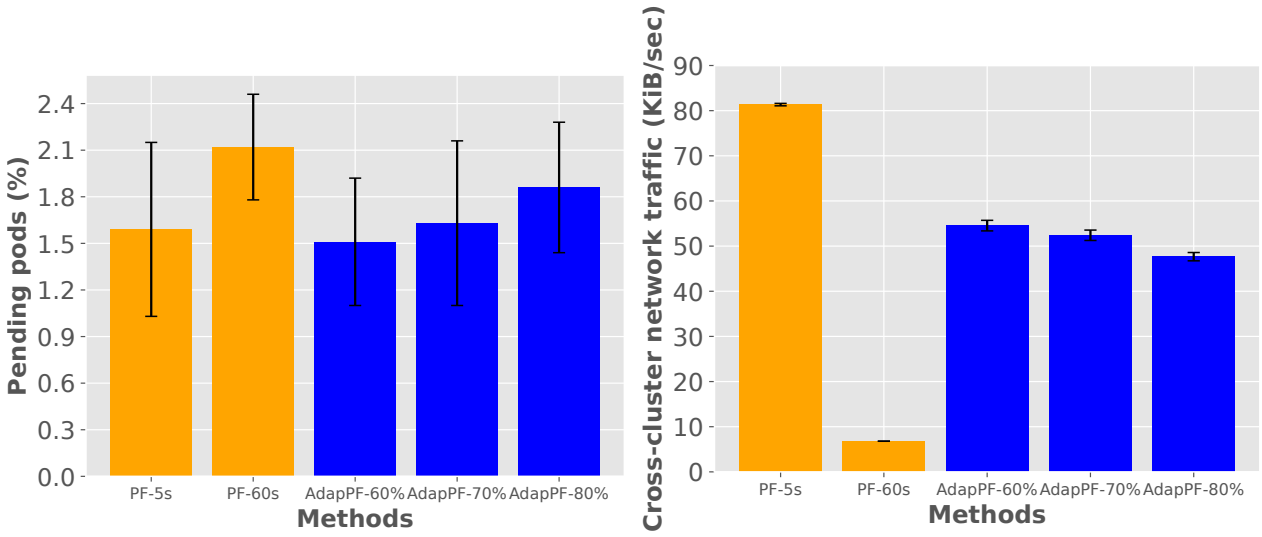
V. Performance Evaluation

We now evaluate the performance of AdapPF using our self-adaptive scrape interval approach.

A. Experimental Setup

We conduct the experiments using the Grid'5000 geo-distributed testbed, which consists of nine server clusters located in various French cities [7].

1) Implementation and experiment deployment: We implement AdapPF-Pullagent in Python 3.10 and follow the design features discussed in Section IV. For the experiment environment, we utilize Kubernetes v1.23.5 as our container orchestration platform and various packages to provide different functionalities, including multi-cluster Kubernetes (mck8s) to distribute workloads across member clusters, Cilium v1.11.4 as the Container Network Interface (CNI), Prometheus v2.34.0, Node-exporter v1.3.1, and Pushgateway v1.5.1. To ensure the robustness of our software, we deploy AdapPF-Pullagent using a Kubernetes Deployment [40]. This setting enables automatic restart of the software in case of component errors.



(a) Percentage of pending pods

(b) Cross-cluster network traffic

Fig. 4. Experiment results of percentage of pending pods (a) and cross-cluster network traffic (b).

In our experiments, we deploy a Kubernetes cluster that serves as the global-view cluster, consisting of two nodes dedicated to the control plane and the worker node. Each node has 4 CPU cores and 16 GiB of memory. We then launch 5 clusters to be our member clusters. Each cluster has nine nodes (one control plane and eight worker nodes). All nodes in each member cluster have 2 CPU cores and 8 GiB of memory. Each cluster incorporates the Prometheus server installation, with the node-exporter deployed in every node.

2) Performance indicators and test methods: This work aims to maintain monitoring data accuracy while reducing cross-cluster network traffic in a geo-distributed cluster federation environment. Therefore, we use resource-based scheduling and measure the percentage of pending pods under an execution workload. Meanwhile, we collect the cross-cluster network traffic using *tcpdump*. A low percentage of pending pods and network traffic indicates better performance.

To evaluate AdapPF's performance, we conduct a comparison with the unmodified Prometheus Federation. We set the Prometheus Federation's scrape interval to 5 s and 60 s and compare against AdapPF's self-adaptive scrape interval approach with different $Rmax_{cluster}$ (60%, 70%, and 80%). For all member clusters, we set $Tmin_{cluster}$ and $Tmax_{cluster}$ to 5 (mck8s default) and 60 (Prometheus default) seconds, respectively. $Rmin_{cluster}$ is set to 0% to reduce the complexity of the experiment. All Prometheus servers have set their scrape interval to 5 seconds. Similar to the experiments from Section II, we use original Google cluster-usage traces as a dataset representing high resource usage. These traces contain information regarding numerous deployed applications, including essential parameters such as resource demands (CPU, RAM), job duration, and job inter-arrival times. This dataset has been extensively utilized for evaluating resource scheduling [41]. We inject workloads (1,095 jobs) for 60 minutes and then wait 30 minutes to release the computing resources. We run each experiment 10 times and present the results in the next section.

B. Experiment Results

Figure 4 illustrates the average and standard deviation results of 10 rounds for the percentage of pending pods (a) and cross-cluster network traffic (b). For clarity in the figures, we use the abbreviation PF to denote Prometheus Federation and AdapPF-60%, AdapPF-70%, and AdapPF-80% to represent AdapPF with different values of $Rmax_{cluster}$.

Figure 4(a) shows that the Prometheus Federation with 60-seconds scrape interval (PF-60s) has the highest average percentage of pending pods (2.12%) whereas PF-5s, AdapPF-60%, AdapPF-70%, and AdapPF-80% respectively have 1.59%, 1.51%, 1.63%, and 1.86%. A greater percentage of pending pods indicates that more pods were scheduled in member clusters which did not have sufficient resources to run them. This situation is because the monitoring data in the management cluster is up to one minute late, which may result in the scheduler scheduling applications to an already fully loaded member cluster. On the other hand, setting a shorter scrape interval leads to a lower percentage of pending pods, which comes at the cost of increased cross-cluster network traffic, as shown in Figure 4(b).

As $Rmax_{cluster}$ is set to 60%, 70%, and 80%, the percentage of pending pods in AdapPF exhibits a progressive increase. This is because AdapPF sets the scrape interval to $Tmin_{cluster}$ when the resources of the target cluster reach $Rmax_{cluster}$. Using Prometheus Federation with 5-seconds scrape interval, the percentage of pending pods is similar to AdapPF-60% and AdapPF-70%. At the same time, by referring to Figure 4(b), we can observe that when comparing the cross-cluster network traffic with PF-5s, both AdapPF-60% and AdapPF-70% can

significantly reduce the traffic from 81.34 KiB/sec to 54.54 KiB/sec and 52.40 KiB/sec, representing a reduction of 33% and 36%, respectively. Despite having lower cross-cluster network traffic, AdapPF-80% yields the greatest percentage of pending pods among the three settings of AdapPF.

This experiment demonstrates that AdapPF can achieve similar accuracy compared to Prometheus Federation with 5-seconds scrape interval while reducing the cross-cluster network traffic between member clusters and management cluster by dynamically adjusting the scrape interval based on the resource usage of target member clusters. We anticipate that the reduction of cross-cluster network traffic will become even more pronounced when deploying more member clusters in a geo-distributed cluster federation environment. Our approach differs from Prometheus Federation in allowing longer scrape interval for member clusters with lower resource usage, which results in reduced network traffic. On the other hand, when a cluster is experiencing high resource usage, AdapPF can dynamically adjust the scrape interval to a shorter value to get timely monitoring data. By leveraging this, the scheduler or system alarm can make informed decisions or trigger earlier alerts. In contrast, Prometheus Federation's fixed scrape interval leads to fixed cross-cluster network traffic regardless of the cluster's current load, which may waste network bandwidth to report the monitoring data. Note that this design may need time to adjust the scrape interval in case of a sudden load surge. The administrator may define an appropriate $Tmax_{cluster}$ based on their system load patterns.

VI. Conclusion

This paper presents Adaptive Prometheus Federation (AdapPF), an extension of Prometheus Federation, designed explicitly for a geo-distributed cluster federation environment. AdapPF uses a self-adaptive approach to dynamically adjust the scrape interval for each member cluster based on the resource status of target clusters. Using actual deployment for experiments, we showed that AdapPF achieves comparable accuracy as Prometheus Federation with 5-seconds scrape interval while reducing cross-cluster network traffic by 36%.

Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[1] C. Wöbker *et al.*, "Fogernetes: Deployment and management of fog computing applications," in *Proc. IEEE/IFIP NOMS*, 2018.
[2] J. Zhang *et al.*, "Kole: Breaking the scalability barrier for managing far edge nodes in cloud," in *Proc. SoCC*, 2022.
[3] Multicluster Special Interest Group, "Kubernetes Cluster Federation," https://bit.ly/3QW2cHw, cited Mar 2023.
[4] M. A. Tamiru *et al.*, "mck8s: An orchestration platform for geo-distributed multi-cluster environments," in *Proc. ICCCN*, 2021.
[5] Prometheus, "Overview," https://reurl.cc/e3A1aL, cited Mar 2023.
[6] Prometheus, "Federation," https://reurl.cc/9Ga2E8, cited Mar 2023.
[7] F. Cappello *et al.*, "Grid'5000: A large scale and highly reconfigurable grid experimental testbed," in *Proc. IEEE/ACM Intl Workshop on Grid Computing*, 2005.
[8] C. Reiss *et al.*, "Google cluster-usage traces: format + schema," Tech. Rep., 2011.
[9] C.-K. Huang and G. Pierre, "Acala: Aggregate Monitoring for Geo-Distributed Cluster Federations," in *Proc. SAC*, 2023.
[10] F. Bonomi *et al.*, "Fog computing and its role in the Internet of Things," in *Proc. MCC*, 2012.
[11] A. J. Fahs *et al.*, "Voilà: Tail-latency-aware fog application replicas autoscaler," in *Proc. MASCOTS*, 2020.
[12] C.-K. Huang *et al.*, "S-cache: Toward an low latency service caching for edge clouds," in *Proc. PERSIST-IoT*, 2019.
[13] C.-K. Huang and S.-H. Shen, "Enabling service cache in edge clouds," *ACM Transactions on Internet of Things*, vol. 2, no. 3, 2021.
[14] P. Souza Jr *et al.*, "Good shepherds care for their cattle: Seamless pod migration in geo-distributed Kubernetes," in *Proc. IEEE ICFEC*, 2022.
[15] J. Huang *et al.*, "RLSK: A job scheduler for federated Kubernetes clusters based on reinforcement learning," in *Proc. IEEE IC2E*, 2020.
[16] D. Kim *et al.*, "TOSCA-based and federation-aware cloud orchestration for Kubernetes container platform," *Applied Sciences*, vol. 9, no. 1, 2019.
[17] B. Shayesteh *et al.*, "Automated concept drift handling for fault prediction in edge clouds using reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, 2022.
[18] J. Povedano-Molina *et al.*, "Dargos: A highly adaptable and scalable monitoring architecture for multi-tenant clouds," *Future Generation Computer Systems*, vol. 29, no. 8, 2013.
[19] D. Trihinas *et al.*, "Jcatascopia: Monitoring elastically adaptive applications in the cloud," in *Proc. CCGrid*, 2014.
[20] Nagios Community, "Nagios," http://www.nagios.org/, cited Mar 2023.
[21] S. A. De Chaves *et al.*, "Toward an architecture for monitoring private clouds," *IEEE Communications Magazine*, vol. 49, no. 12, 2011.
[22] P. Tader, "OpenNebula: A cloud management tool," *Linux Journal*, vol. 2010, no. 195, 2010.
[23] B. Costa *et al.*, "Monitoring fog computing: A review, taxonomy and open challenges," *Computer Networks*, vol. 215, 2022.
[24] Á. Brandón *et al.*, "FMonE: A flexible monitoring solution at the edge," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.
[25] S. Forti *et al.*, "Lightweight self-organising distributed monitoring of fog infrastructures," *Future Generation Computer Systems*, vol. 114, 2021.
[26] M. Großmann and C. Klug, "Monitoring container services at the network edge," in *Proc. ITC*, 2017.
[27] A. Aznavouridis *et al.*, "Micro-service placement policies for cost optimization in Kubernetes," in *Proc. ANIA*, 2022.
[28] G. Carcassi *et al.*, "SLATE: Monitoring distributed Kubernetes clusters," in *Proc. ACM PEARC*, 2020.
[29] Y.-W. Chan *et al.*, "Implementation of a cluster-based heterogeneous edge computing system for resource monitoring and performance evaluation," *IEEE Access*, vol. 10, 2022.
[30] T. Hu and Y. Wang, "A Kubernetes autoscaler based on pod replicas prediction," in *Proc. ACCTCS*, 2021.
[31] T. Lin *et al.*, "Towards an end-to-end network slicing framework in multi-region infrastructures," in *Proc. IEEE NetSoft*, 2020.
[32] L. Toka *et al.*, "Machine learning-based scaling management for Kubernetes edge clusters," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, 2021.
[33] D. Battulga *et al.*, "LivingFog: Leveraging fog computing and LoRaWAN technologies for smart marina management (experience paper)," in *Proc. ICIN*, 2022.
[34] J. Cho and Y. Kim, "A design of serverless computing service for edge clouds," in *Proc. ICTC*, 2021.
[35] P. Trakadas *et al.*, "Scalable monitoring for multiple virtualized infrastructures for 5G services," in *Proc. SoftNetworking*, 2018.
[36] T. Dockendorf *et al.*, "Early experiences with tight integration of Kubernetes in an HPC environment," in *Proc. PEARC*, 2022.
[37] V. Colombo *et al.*, "Towards self-adaptive peer-to-peer monitoring for fog environments," in *Proc. SEAMS*, 2022.
[38] Prometheus, "node-exporter," https://github.com/prometheus/node_exporter, cited Mar 2023.
[39] Prometheus, "pushgateway," https://github.com/prometheus/pushgateway, cited Mar 2023.
[40] Kubernetes, "Deployments," https://reurl.cc/9pgGkx, cited Mar 2023.
[41] C. Reiss *et al.*, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. Socc*, 2012.