Scalability of public geo-distributed fog computing federations

Chih-Kai Huang

COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Informatique*

Par
Chih-Kai HUANG

Scalability of Public Geo-Distributed Fog Computing Federations

Thèse présentée et soutenue à Rennes, le 09 Décembre 2024
Unité de recherche : IRISA (UMR 6074)

Rapporteurs avant soutenance :

Romain ROUVOY Professeur des Universités, Université de Lille
Pierre SENS Professeur des Universités, Sorbonne Université

Composition du Jury :

Président :	Anne-Cécile ORGERIE	Directrice de Recherche, CNRS
Examinateurs :	Ivona BRANDIĆ	Professeure, Technische Universität Wien
	Romain ROUVOY	Professeur des Universités, Université de Lille
	Pierre SENS	Professeur des Universités, Sorbonne Université
	Monica VITALI	Professeure associée, Politecnico di Milano
Dir. de thèse :	Guillaume PIERRE	Professeur des Universités, Université de Rennes

Experiments presented in Chapter 4 and Chapter 5 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).

Acknowledgement

First of all, I would like to express my sincere gratitude to my supervisor, Professor Guillaume Pierre, for his continuous guidance and strong support in my research and life throughout my Ph.D. journey in France. This journey would not be delightful and smooth without his patience and guidance. It is a great honor to be supervised by such an outstanding professor. I will continue to help others as he has helped me. I would also like to thank Dr. Shadi Ibrahim for his help at the very beginning of this thesis.

I would like to thank my jury members: Professor Romain Rouvoy, Professor Pierre Sens, Professor Ivona Brandić, Dr. Anne-Cécile Orgerie, and Professor Monica Vitali for taking the time to review and evaluate my thesis and for giving me valuable feedback and discussions.

I am also deeply grateful to CSID committee members: Professor Erik Elmroth and Professor Cédric Tedeschi for their invaluable feedback and support over these years.

Many thanks to all the Magellan (Myriads) team members for their support, chats, and discussions. Special thanks to Wedan-Emmanuel Gnibga, Amandine Seigneur, Matthieu Simonin, and Stéphanie Gosselin Lemaile for invaluable help in all aspects. My deepest thanks also go to CMI Rennes for providing excellent services and taking care of all my complex administrative procedures in France. I sincerely thank the many wonderful people I have met during these years: Shengnan Yao, Zhilei Luo, Kai Gu and more.

My deepest gratitude to my incredible family for their invincible support, care, and love that makes this Ph.D. come true. This achievement is not only for me but also for my family. To see my family proud would be my greatest accomplishment.

Finally, I would like to give myself a big thumbs up for all the work I have put into my Ph.D. over the past three years. I am proud of myself and excited to see what comes next.

RÉSUMÉ

Ces dernières années, le cloud computing est devenu une technologie importante offrant une gamme d'avantages par rapport aux déploiements sur site, tels que l'évolutivité, le faible coût et la flexibilité. La nature publique, partagée et à la demande des plateformes de cloud computing les rend accessibles à d'innombrables utilisateurs allant des individus et petites entreprises aux grandes entreprises. Cela simplifie les tâches liées à l'informatique pour les utilisateurs, qui n'ont alors qu'à se concentrer sur leur activité principale sans avoir à construire et à maintenir leur propre infrastructure informatique. Les utilisateurs peuvent facilement développer et déployer leurs applications sur les plateformes de cloud grâce à une grande variété d'outils et de services maintenus par les Cloud Service Providers (CSPs). Une plateforme de cloud public comprend généralement un petit nombre de centres de données centralisés, chacun incluant un grand nombre de serveurs de calcul, de capacité réseau et de ressources de stockage.

Lorsque les utilisateurs déploient leurs applications sur une plateforme de cloud public, ils prennent en compte différentes exigences non fonctionnelles pour chaque application, telles que la capacité de ressources nécessaire, la haute disponibilité et la tolérance aux pannes. L'émergence d'applications sensibles à la latence, comme la réalité virtuelle et le streaming vidéo à 360 degrés, crée de nouvelles exigences telles qu'une faible latence réseau de bout en bout. En outre, le développement rapide de l'Internet des Objets (IoT) a conduit à une augmentation spectaculaire du nombre de dispositifs IoT, qui génèrent une énorme quantité de données devant être envoyées aux centres de données cloud distants pour un traitement des données en temps réel.

Bien que l'expansion mondiale des centres de données cloud et les améliorations significatives de l'accessibilité au réseau et de la capacité de bande passante aient réduit la latence entre les utilisateurs finaux et leurs applications cloud, la centralisation des grands centres de données signifie qu'ils restent physiquement éloignés de la plupart des utilisateurs finaux. Une grande latence de bout en bout pour les applications et la transmission longue distance de grandes quantités de données rendent les applications sensibles à la latence et certaines applications IoT inadaptées aux environnements cloud traditionnels.

Pour répondre à leurs exigences et pallier les limitations des plateformes cloud traditionnelles, le fog computing est apparu comme une solution viable.

Le fog computing ne vise pas à remplacer l'utilisation du cloud computing. Il est plutôt conçu comme une extension du cloud computing. Le concept de base est de rapprocher les ressources de calcul du bord du réseau pour combler l'écart du continuum numérique entre les utilisateurs et les plateformes de cloud. En servant les utilisateurs et en traitant les données à proximité, le fog computing a le potentiel de réduire considérablement la latence de bout en bout, améliorant ainsi la qualité de l'expérience utilisateur. Cette caractéristique est cruciale pour les applications nécessitant des interactions en temps réel. De même, les plateformes de fog peuvent également optimiser la quantité de données transmises sur de longues distances et n'envoyer que les données les plus importantes et pertinentes aux centres de données distants pour traitement, tandis que d'autres données peuvent être traitées localement dans la plateforme de fog.

La plupart des solutions actuelles de fog computing, qu'elles soient commerciales ou open-source, exigent que les utilisateurs de fog construisent leur propre plateforme de fog privée dans la zone pertinente avec du matériel et des logiciels dédiés. Cependant, cette pratique annule les économies d'échelle réalisées grâce au partage multi-tenant et au multiplexage statistique du cloud, et ramène les services à un modèle pré-cloud où chaque application nécessitait une mise en service individuelle avec une infrastructure personnalisée. De plus, ces solutions nécessitent un investissement initial élevé et ne permettent pas d'ajuster facilement à la hausse ou à la baisse les ressources provisionnées, ce qui pose des défis importants aux utilisateurs individuels et aux startups, ralentissant l'adoption des technologies de fog computing.

Pour permettre à tous les types d'utilisateurs allant des particuliers aux petites entreprises et aux grandes entreprises, de bénéficier d'avantages tels qu'une faible latence de bout en bout, une disponibilité des ressources à la demande et une flexibilité, il est crucial de développer des plateformes de fog computing publiques, multi-tenant, géo-distribuées à grande échelle qui peuvent couvrir un pays ou même un continent entier. Les utilisateurs pourront tirer parti de cette plateforme de fog publique pour développer des logiciels et déployer des applications, ce qui permettra de réduire l'investissement massif dans l'infrastructure nécessaire pour installer une plateforme de fog privée et leur permettre de se concentrer sur leurs activités principales et d'innover continuellement sans le fardeau de la gestion des infrastructures de fog.

Malgré le grand potentiel des futures plateformes de fog computing publiques à grande échelle, il n'existe actuellement aucune plateforme de ce type sur le marché. Construire une plateforme publique à grande échelle pose plusieurs défis, comme des considérations économiques. Par exemple, il faudrait un investissement important pour installer un nombre suffisant de matériel et de logiciels de fog. Le modèle de profit et d'affaires doit encore être étudié en raison de la nouveauté de ce concept. En outre, d'innombrables détails scientifiques et techniques doivent encore être résolus, tels que l'évolutivité, l'automatisation, la sécurité et la durabilité.

Cette thèse explore spécifiquement trois défis clés liés à l'évolutivité des futures plateformes de fog computing publiques géo-distribuées. Premièrement, une seule entreprise aurait du mal à déployer un nombre suffisant de ressources de fog pour couvrir une vaste zone géographique tout en attirant suffisamment de charges de travail pour générer une haute utilisation des ressources, en particulier pour les ressources de fog situées dans les zones rurales. La plus faible densité de population dans ces régions pourrait entraîner une sous-utilisation des ressources de fog, ce qui augmenterait à son tour le coût global pour un fournisseur de fog. Il est donc nécessaire de concevoir un modèle qui puisse répondre à la fois à la couverture de service et à l'utilisation des ressources.

Deuxièmement, une plateforme de fog publique géo-distribuée à grande échelle peut devoir desservir un grand nombre d'utilisateurs simultanément, ce qui nécessite donc un cadre d'orchestration robuste pour gérer les ressources nécessaires. Ce cadre doit donc être capable de gérer de nombreux utilisateurs, charges de travail et dispositifs de fog de manière efficace. De plus, la nature partagée d'une plateforme de fog publique signifie que ce système doit inclure des méthodes robustes de gestion de la multi-location pour isoler les utilisateurs et les empêcher de se nuire mutuellement.

Troisièmement, la surveillance est une fonctionnalité essentielle pour mesurer l'utilisation des ressources dans les environnements de calcul modernes, identifier les pannes potentielles et héberger efficacement les applications. Cette capacité est particulièrement cruciale dans les plateformes de fog à grande échelle, géo-distribuées et potentiellement instables. Cependant, surveiller une grande plateforme géo-distribuée est difficile car les données de surveillance proviennent d'un grand nombre d'infrastructures de fog distribuées et doivent être transmises sur de longues distances. Le trafic réseau causé par la surveillance peut gaspiller les liaisons réseau existantes et finir par représenter une quantité importante du trafic de gestion du système. Il est donc important d'étudier des méthodes de

surveillance efficaces pour équilibrer le volume de trafic réseau et la précision des données à grande échelle.

Cette thèse propose une série de solutions pour relever les défis d'évolutivité mentionnés ci-dessus. Nous tirons parti du concept de cluster federation comme solution pour concevoir une plateforme de fog à grande échelle qui déploie stratégiquement plusieurs clusters géo-distribués et qui peut être gérée et utilisée comme un seul cluster homogène. Nos contributions s'appuient sur les écosystèmes populaires d'orchestration de conteneurs Kubernetes et de système de surveillance Prometheus. Cependant, nous avançons que les principes et algorithmes introduits dans cette thèse peuvent s'adapter et s'intégrer à d'autres solutions d'orchestration de conteneurs et systèmes de surveillance existants et futurs.

Première Contribution : Gestion Multi-Tenant dans les Méta-Fédération Fog Scalables

La première contribution de cette thèse aborde les problèmes liés à la couverture des services et à l'utilisation des ressources. En raison de la difficulté à déployer un nombre suffisant de clusters de fog à travers un pays ou un continent, nous supposons que plusieurs petits ou moyens fournisseurs de services de fog, chacun dans une région différente, peuvent s'associer pour créer une plateforme à grande échelle en utilisant une féfération de clusters fog. Dans ce contexte, nous proposons le concept de *méta-fédérations*, où des fournisseurs locaux de fog indépendants peuvent louer de manière flexible leurs clusters fog les uns aux autres. En appliquant cette idée, un fournisseur de fog peut utiliser des clusters fog d'autres régions, opérés par différents fournisseurs de fog, afin d'étendre la couverture de service dans des zones où il ne possède pas lui-même de ressources. De plus, les clusters fog situés dans des zones à faible densité peuvent être loués à plusieurs fournisseurs de services de fog afin d'augmenter l'utilisation des ressources et ainsi réduire le coût global pour les propriétaires des clusters.

La mise en œuvre du concept de grandes méta-fédérations, dans lesquelles des milliers de fournisseurs fog locaux louent leurs clusters à des centaines de fédérations indépendantes, nécessite de relever deux principaux défis. (i) Multi-Tenancy : dans le cadre de la conception des meta-federations, chaque cluster peut être partagé par différentes fédérations. Les applications dans le même cluster créées par des utilisateurs de différentes fédérations ne doivent pas pouvoir interférer les unes avec les autres. Une fog federation peut également inclure un grand nombre d'utilisateurs. Les charges de travail soumises par

différents utilisateurs au sein d'une fédération doivent également bénéficier de garanties d'isolation similaires. (ii) Scalabilité : cette plateforme de fog à grande échelle et distribuée géographiquement peut inclure de nombreux clusters fog. Par conséquent, chaque féfération doit être capable de gérer un grand nombre de clusters membres, tandis que chaque cluster membre doit pouvoir louer ses ressources à un grand nombre de fédérations.

Pour établir une base solide pour le développement de futures plateformes publiques de fog computing multi-tenant à grande échelle et relever les défis introduits par les meta-federations, cette thèse présente UnBound, une plateforme de meta-federations de fog scalable. UnBound exploite le framework d'orchestration de conteneurs Kubernetes pour gérer les ressources au sein de chaque cluster de fog et Open Cluster Management (OCM) pour fédérer plusieurs clusters membres sous la gouvernance centralisée d'un cluster de gestion. OCM est un orchestrateur open-source et extensible, spécialement conçu pour Kubernetes dans des scénarios multi-clusters, qui prend en compte la scalabilité d'une fédération. UnBound aborde les problèmes de gestion multi-tenant en utilisant le projet Virtual Kubernetes Clusters (vCluster). UnBound l'utilise pour créer des sous-clusters logiques afin d'isoler les fédérations au sein d'un cluster membre. Chaque vCluster possède son propre serveur API et son propre magasin de données, ce qui garantit une isolation stricte. En ce qui concerne les utilisateurs d'une fédération, nous utilisons les Namespaces de Kubernetes pour les isoler grâce à la fonctionnalité de vCluster qui permet aux utilisateurs de créer des ressources à l'échelle du cluster.

Des évaluations avec des fédérations de jusqu'à 500 clusters Kubernetes distribués géographiquement démontrent que UnBound maintient des temps de déploiement d'applications comparables à ceux de l'Open Cluster Management original dans un seul cluster membre, évite l'augmentation du trafic réseau entre les clusters, maintient la consommation de ressources dans des limites acceptables, et montre stabilité et scalabilité, en faisant une solution adaptée pour des déploiements de fog computing à grande échelle.

Deuxième Contribution : Systèmes de Surveillance Efficaces des Fédération Fog géo-distribuées

La surveillance distribuée est une fonctionnalité essentielle qui permet aux grandes fédérations de clusters de planifier efficacement le déploiement des applications sur un ensemble de clusters fog géo-distribués. Cela nécessite un système de surveillance robuste tel que Prometheus et son extension Prometheus Federation pour fournir les données de surveillance. Cependant, Prometheus collecte toujours l'état de *tous les serveurs*

disponibles des clusters cibles à une fréquence *fixe*, ce qui peut gaspiller la bande passante réseau dans la fédération tout en étant inutile pour garantir un ordonnancement précis et non scalable avec l'augmentation du nombre de serveurs.

Cette thèse propose deux systèmes de surveillance, Acala et AdapPF, pour résoudre les problèmes de surveillance susmentionnés dans une fédération de clusters Kubernetes géo-distribués. Les deux solutions sont basées sur l'écosystème de surveillance open-source bien connu Prometheus et introduisent des solutions pour équilibrer le trafic réseau inter-clusters et la précision des données de surveillance. Acala vise à fournir au cluster de gestion des informations agrégées sur l'ensemble du cluster plutôt que sur des serveurs individuels, ce qui élève la vue traditionnelle de la surveillance dans Prometheus Federation du niveau « nœud » au niveau « cluster ». De son côté, AdapPF vise à ajuster dynamiquement la fréquence de collecte des données de surveillance pour chaque cluster en fonction de l'état d'utilisation des ressources du cluster.

Nous effectuons des évaluations approfondies des deux systèmes de surveillance à l'aide de déploiements réels dans le banc d'essai géo-distribué Grid'5000. Les résultats montrent que Acala améliore considérablement les performances par rapport à Prometheus. Acala réduit le trafic réseau inter-clusters jusqu'à 97% et diminue la durée de collecte des données jusqu'à 55% dans des expériences sur des clusters à un seul membre. Des expériences plus larges avec jusqu'à 1 000 serveurs montrent qu'il réduit le trafic réseau global d'environ 95%. De plus, nous démontrons que notre solution a un impact minimal sur l'efficacité de l'ordonnancement. L'autre système, AdapPF, atteint une précision de planification comparable à Prometheus Federation avec un intervalle de collecte fixe de 5 secondes tout en réduisant le trafic réseau inter-clusters jusqu'à 36%.

Nous soutenons que les concepts proposés dans Acala et AdapPF peuvent en principe être combinés pour améliorer les performances et l'efficacité du système de surveillance, car ils traitent différentes parties de l'architecture de Prometheus Federation.

Ces contributions fournissent une base solide pour le développement de plateformes de fog computing multi-tenant, publiques, à grande échelle et géo-distribuées, et pour démocratiser les technologies de fog computing.

Abstract

In recent years, cloud computing has become a significant and successful technology that offers a range of advantages over on-premise deployments, such as scalability, affordability, and flexibility. The public, shared, and on-demand nature of cloud computing platforms makes them available to countless users ranging from individuals and small businesses to large enterprises. This keeps IT-related tasks simple for users so that they only need to focus on their core business without having to build and maintain their own IT infrastructure. Users can easily develop and deploy applications on cloud platforms using a wide variety of tools and services maintained by Cloud Service Providers (CSPs). A public cloud platform usually comprises a limited number of centralized data centers, each of which includes a large number of computing servers, network capacity, and storage resources.

When users deploy their cloud applications on a public cloud platform, they consider different non-functional requirements for each application, such as necessary resource capacity, high availability, and fault tolerance. The emergence of latency-sensitive applications such as virtual reality and 360-degree video streaming creates new demands such as low end-to-end network latency. Moreover, the rapid development of the Internet of Things (IoT) has led to a dramatic increase in the number of IoT devices, which generate a massive amount of data that must be sent to remote cloud data centers for real-time data processing.

Although the global expansion of cloud data centers and significant improvements in network accessibility and bandwidth capacity have reduced the latency between the end users and their cloud applications, the centralization of large data centers means they remain physically distant from most end users. Large end-to-end latency for applications and long-distance transmission of large amounts of data make some latency-sensitive and IoT applications unsuitable for traditional cloud environments. To fulfill their requirements and address the limitations of traditional cloud platforms, fog computing has emerged as a viable solution.

Fog computing does not aim to replace the use of cloud computing. Instead, it is designed as an extension of cloud computing. The core concept is to bring computing

resources to the network edge to bridge the computing gap between users and the cloud platforms. By serving users and processing data in closer proximity, fog computing has the potential for significantly reducing end-to-end latency, thereby enhancing the user Quality-of-Experience (QoE). This characteristic is critical for applications that require real-time interactions. Moreover, fog platforms can also optimize the amount of data transmitted over long distances and only send the most important and relevant data to remote data centers for processing, while other data could be processed locally in the fog platform.

Most current fog computing solutions, whether commercial or open-source, require fog users to build their own private fog platform in the designated area with dedicated hardware and software. However, this practice negates the cost efficiencies achieved through the multi-tenancy and statistical multiplexing of cloud computing, and it reverts the services back to a pre-cloud model where each application needed individual provisioning with a custom infrastructure. These solutions also demand high upfront investment and are unable to easily scale up and down the provisioned resources, which makes individual users and startups face significant challenges, slowing down the adoption of fog computing technologies.

To allow all kinds of users, including individuals, small businesses, and large enterprises, to gain advantages such as low end-to-end latency, on-demand resource availability, and flexibility, it is crucial to develop large-scale, public, multi-tenant, geo-distributed fog computing platforms that can cover a country or even an entire continent. Users will be able to leverage this public fog platform to develop software and deploy applications, which in turn can reduce the massive infrastructure investment of preparing a private fog platform and enable them to focus on their core business activities and continuously innovate without the burden of managing fog infrastructures.

Despite the great potential of future large-scale public fog computing platforms, there are currently no such platforms on the market. Building a large-scale public platform faces several challenges such as economic considerations. For example, it would require a significant investment to install enough number of fog hardware and software. The profit and business model still needs to be investigated due to the novelty of this computing concept. In addition, countless scientific and technical details should still be addressed, such as scalability, automation, security, and sustainability.

This thesis specifically explores three key challenges related to the scalability of future public geo-distributed fog computing platforms. First, a single company would find it

challenging to deploy enough number of fog resources to cover a large geographic area while attracting sufficient workloads to generate high resource utilization, especially for fog resources located in rural areas. The smaller population density in these regions may result in the underutilization of fog resources, which may in turn increase the overall cost for a fog provider. Therefore, it is necessary to design a model that can address service coverage and resource utilization at the same time.

Second, a large-scale public geo-distributed fog platform may need to serve a large number of users simultaneously, which therefore demands a robust orchestration framework to manage the necessary resources. The framework must therefore be able to handle numerous users, workloads, and fog devices efficiently. Moreover, the shared nature of the public fog platform means that this framework must include robust multi-tenancy management methods to isolate users and prevent them from interfering with each other.

Third, monitoring is an essential functionality to track resource usage of modern computing environments, identify potential failures, and efficiently schedule applications. This capability is particularly critical in large-scale, geo-distributed, and unstable fog platforms. However, monitoring a large geo-distributed platform is difficult because the monitoring data come from a large number of distributed fog infrastructures and need to be transmitted over long distances. The network traffic caused by monitoring may waste the existing network links and may eventually account for a significant amount of the system management traffic. Therefore, it is important to investigate efficient monitoring methods to balance the volume of network traffic and data accuracy on a large scale.

This thesis proposes a series of solutions to address the scalability challenges mentioned above. We leverage the concept of cluster federation as a solution to design a large-scale fog platform that strategically deploys multiple geo-distributed clusters and can be managed and used as a single homogeneous cluster. Our contributions are based on the popular Kubernetes container orchestration and the Prometheus monitoring framework ecosystems. However, we argue that the principles and algorithms introduced in this thesis can adapt and integrate seamlessly with existing and future container orchestration solutions and monitoring systems.

Contribution 1: Multi-Tenancy Management in Scalable Fog Meta-Federations

The first contribution of this thesis addresses the issues related to service coverage and resource utilization. Due to the difficulty of deploying enough fog clusters across a country or a continent, we assume that several small or medium-sized fog service providers, each

of which in a different region, may team up to deliver a large-scale platform using a fog cluster federation. In this context, we propose the concept of *meta-federations*, where independent local fog providers can flexibly lease their own fog clusters to one another. By applying this idea, a single fog provider may use fog clusters from other regions that are operated by different fog providers to expand service coverage in locations where they do not own resources themselves. Moreover, fog clusters located in low-density areas may be leased to multiple fog service providers to increase resource utilization and thereby reduce the overall cost for cluster owners.

Implementing the concept of large meta-federations in which thousands of local fog providers rent their fog clusters to hundreds of independent federations requires one to address two main challenges. (i) Multi-Tenancy: As part of the meta-federations design, each cluster may be shared by different federations. Applications in the same cluster created by users from different federations should not be able to interfere with each other. A fog federation may also include a large number of users. Workloads submitted by different users in a federation should also have similar isolation guarantees. (ii) Scalability: This large-scale geo-distributed fog platform may include many fog clusters. Therefore, each management cluster should be able to handle a large number of member clusters, while each member cluster should be able to lease its resources to a large number of management clusters.

To establish a cornerstone for developing future large-scale, public, multi-tenant fog computing platforms and address the challenges introduced by meta-federations, this thesis presents UnBound, a scalable fog meta-federations platform. UnBound leverages the Kubernetes container orchestration framework to manage resources within each fog cluster and Open Cluster Management (OCM) to federate multiple member clusters under the centralized governance of a management cluster. OCM is an open-source, extensible orchestrator specifically designed for Kubernetes in multi-cluster scenarios, which takes into account the scalability of a federation. UnBound addresses the multi-tenancy management issues by leveraging the Virtual Kubernetes Clusters (vCluster) project. UnBound uses it to create logical sub-clusters to isolate the federations within a member cluster. Each vCluster has its own API server and data store, which provides hard isolation guarantees. As for the users in a federation, we use Kubernetes Namespaces to isolate them thanks to the vCluster functionality that allows users to create cluster-scoped resources.

Comprehensive evaluations with federations of up to 500 geo-distributed Kubernetes clusters demonstrate that UnBound maintains comparable application deployment times

to the original Open Cluster Management in a single member cluster, avoids increasing cross-cluster network traffic, keeps resource consumption within acceptable boundaries, and exhibits stability and scalability, making it a suitable solution for large-scale fog computing deployments.

Contribution 2: Efficient Monitoring Frameworks in Geo-Distributed Cluster Federations

Distributed monitoring is an essential functionality that allows large cluster federations to efficiently schedule applications on a set of available geo-distributed fog clusters. This requires a robust monitoring framework such as Prometheus and its extension Prometheus Federation to provide the monitoring data. However, Prometheus always collects the status of *every available server* from target clusters at a *fixed* frequency, which may waste network bandwidth in the federation while being unnecessary for ensuring accurate scheduling and unscalable with increasing server number.

This thesis proposes two monitoring frameworks, Acala and AdapPF, to address the above monitoring issues in a geo-distributed Kubernetes cluster federation. Both solutions are based on the well-known open-source Prometheus monitoring ecosystem and introduce solutions to balance cross-cluster network traffic and the accuracy of monitoring data. Acala aims to provide the management cluster with aggregate information about the entire cluster instead of individual servers, which elevates the traditional view of monitoring in Prometheus Federation from the "node" level to the "cluster" level. AdapPF aims to dynamically adjust the collection frequency of monitoring data for each cluster based on the resource utilization status of the cluster.

We perform extensive evaluations of both monitoring frameworks using actual deployments in the geo-distributed Grid'5000 testbed. The results show that Acala achieves significant performance improvements compared to traditional Prometheus. Acala reduces cross-cluster network traffic by up to 97% and decreases scrape duration by up to 55% in single-member cluster experiments. Larger experiments with up to 1,000 servers demonstrate that it reduces the overall network traffic by about 95%. Moreover, we demonstrate that our solution has minimal impact on scheduling efficiency. The other framework, AdapPF, achieves comparable scheduling accuracy to Prometheus Federation with a fixed 5 seconds scrape interval while reducing cross-cluster network traffic by up to 36%.

We argue that the concepts proposed in Acala and AdapPF can in principle be combined together to improve monitoring system performance and efficiency, considering that they address different parts of the Prometheus Federation architecture.

These contributions provide a solid foundation for developing future large-scale, public, multi-tenant, geo-distributed fog computing platforms and democratizing fog computing technologies.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing, as one of the most successful computing paradigms, has revolutionized the way enterprises develop software and deploy applications. Traditionally, enterprises had to invest money in building their own server rooms with Information Technology (IT) devices, such as servers, switches, and firewalls. Nowadays, by using cloud resources from public cloud providers with high performance, flexibility, on-demand resource availability, reliability, and scalability, enterprises can focus on their major business and other priorities without worrying about IT-related installation and maintenance [1]. In addition, the public and shared nature of cloud computing platforms makes them available to numerous users ranging from individuals and small companies to large enterprises.

Cloud Service Providers (CSPs) typically build and maintain their own cloud data centers, which are composed of a large number of computing servers, network devices, and storage resources. To improve resource utilization, virtualization technologies enable CSPs to abstract cloud resources and share them with multiple users without letting them interfere with each other. For example, a single physical server can run multiple Virtual Machines (VMs) using hardware virtualization technology [2].

The number of data centers for each CSP is typically small. For instance, Figure 1.1 shows the regions of Google Cloud, which includes 40 locations in the world and plans to launch a few new regions such as Mexico [3]. This situation causes data centers to be often physically distant from the end users, possibly resulting in high network access latency [4].

According to a survey from the Enterprise Strategy Group, there is a strong trend toward multi-cloud application deployment [5]. The findings indicate that 85% of organizations leverage two or more CSPs for their deployments. One of the key reasons for this trend is that applications can deploy and distribute to different data center locations to gain the advantages of multi-cloud deployments, such as reducing the user-to-cloud latency [6]. This indicates a need to increase the number of locations where cloud tenants can deploy their applications.

Figure 1.1 – A map of Google Cloud regions around the world. Blue dots show current regions. Triangle represents future regions [3].

The emergence of latency-sensitive applications requires reducing the user-to-cloud latency, which may not be compatible with traditional cloud computing deployments [7]. For instance, head-tracking applications such as virtual reality and 360-degree video streaming request that network transmission combined with application processing times should remain under 20 milliseconds to avoid motion sickness [8]. Another demanding use case is the Internet of Things (IoT), which is experiencing rapid growth with 127 additional devices being connected to the Internet every second [9]. The number of connected IoT devices worldwide in 2030 is predicted to be greater than 32 billion [10]. Typically, these IoT devices generate data that should be sent to a cloud data center for real-time processing due to the limited computing capabilities of the IoT devices themselves [11]. Moreover, the long-distance data transmission of large amounts of data over the network may eventually saturate the existing network links [12].

To address the limitations of data-center-based cloud computing architectures, an intuitive solution is to place the computing resources closer to the end users. Fog computing proposes such an extension of cloud computing that distributes computing resources at

the network edge, close to the end users, and to the location of data sources generated by IoT devices [13]. This design enables computational tasks to be performed close to the data sources, reducing end-to-end latency to improve the user Quality-of-Experience (QoE) and reducing the volume of data transmitted to data centers.

Since the introduction of the fog computing concept in 2012 [13], various fog/edge computing solutions have been developed and made available on the market. For example, Microsoft provides Azure Stack Edge, a Hardware-as-a-Service (HaaS) product that allows customers to order Azure-managed devices and deploy them in the desired locations [14]. These hardware devices contain the computing, storage, and intelligent capabilities of the Azure cloud and can process the data directly at the network edge. Another similar solution called "Google Distributed Cloud connected" is designed for customers who want to obtain real-time insights from data at the local level with low latency [15]. This solution offers a flexible selection of hardware options that allow customers to select and build the right-sized fog/edge infrastructure locally. Other CSPs propose similar fog/edge solutions, such as Oracle Roving Edge Infrastructure [16] and Amazon Web Services Snowball Edge Compute [17]. In the open-source world, several solutions also exist. The FogGuru project develops the LivingFog platform, which allows users to leverage Raspberry Pi clusters to process IoT data transmitted with the LoRa long-distance wireless protocol [18]. This platform has been successfully used for smart water management and smart-city data processing [19]. KubeEdge is a Kubernetes-based framework that aims to bring container orchestration on devices at the network edge [20]. KubeEdge's architecture includes two main components: CloudCore and EdgeCore. Users deploy CloudCore in the cloud to manage the edge devices and install EdgeCore in user-owned edge devices that may be placed in strategic edge locations such as factories.

Considering the current fog/edge solutions, deploying applications close to the end users to benefit from the advantages of fog computing requires one to deploy customized hardware infrastructures and/or software at appropriate locations to form a private fog deployment. However, assigning a specific set of devices to accommodate a single use case within a particular location weakens the economies of scale delivered by the multi-tenancy and statistical multiplexing principles of cloud computing. This situation brings fog computing back to a pre-cloud era where each application required dedicated hardware to be provisioned, limiting its geographical scope to a small choice of locations where suitable devices have been deployed. Moreover, this may lead to high costs and large delays in setting up these hardware devices, which in turn could slow down innovation.

As a result, many potential fog computing users may thereby be unable to fully utilize fog technology due to these technical and management limitations. In terms of flexibility, as their business expands, scaling up fog infrastructures may prove to be difficult and time-consuming. These issues show that the current fog computing solutions do not fully enable the potential and unique capabilities of fog computing.

Fog computing was initially designed as an extension of cloud computing. We therefore believe it should follow the same guiding principles as cloud computing. In particular, users around the world who are unable to build their own private fog platform should be able to deploy their applications in a public fog computing platform with zero upfront infrastructure cost. Enabling fog computing to embrace the full benefits of cloud computing principles requires the design of *large-scale*, *public*, *multi-tenant*, geo-distributed fog computing platforms that can cover a whole country or even a continent. Similar to public cloud platforms, public fog platforms should exploit the statistical multiplexing of large numbers of independent workloads to help guarantee high resource utilization and therefore reduce the cost of building and maintaining the fog platform. In doing this, public fog platforms may help unlock the full potential of fog computing that makes computing resources accessible to all kinds of users, letting them focus on their major affairs and pursue innovation. Moreover, with the rapid growth in the number of IoT devices and the increasing demand for real-time data processing, we believe it is particularly important to design large-scale public shared fog computing platforms.

Unfortunately, although fog computing technology has demonstrated its potential and advantages in many fields, no large-scale public fog platforms exist today that are easily accessible to any user. There are many reasons for this, ranging from economic aspects to technical and scientific ones. On the economic side, building a public fog platform would require large investments to deploy the fog infrastructures, develop the software, and maintain and operate the platform. Compared to cloud computing, the concept of fog computing is relatively new. Therefore, the revenue or service model still needs to be further investigated to ensure that the Return on Investment (ROI) is within an acceptable range. In addition, environmental protection and data regulatory challenges also make it difficult to design a compliant large-scale platform due to its decentralized nature. A very large fog platform may cover different countries, which means that these distributed fog infrastructures would need to pass regional environmental assessments and follow complex data protection laws in different countries.

The scientific aspects of constructing large-scale fog platforms are also very challenging. A large number of fog computing servers and user's applications must be distributed in various strategic locations, which makes management and monitoring face stringent scalability issues. Moreover, maintaining a platform manually is costly, which encourages developing robust automation methods to remotely maintain and repair these physical servers and network devices. Applications deployed by users also need to take into account automatic scheduling, migration, and failover. Compared to centralized data centers with co-located hardware that can be protected together, guaranteeing security is clearly a difficult issue in decentralized fog architectures, which may be more vulnerable than centralized ones to physical or network attacks. Sustainability can also be a key challenge for such platforms, and energy efficiency should be considered to keep the fog clusters running with a balance between energy consumption and speed of computation, which thereby can help minimize operational costs and keep platforms effective [21]. As a result, these challenges make building large-scale fog platforms difficult, which requires a series of solutions for them.

This thesis addresses the challenges of designing scalable public fog computing infrastructures, with the aim to eventually be able to cover an entire country or even a continent. Within this scope, we specifically focus on three main issues whose solutions may constitute a basis for further research toward the design of future large-scale, public, multi-tenant, geo-distributed fog computing platforms.

The first issue addressed in this thesis is to enable broad service coverage of a fog resource provider. Deploying a sufficient number of geo-distributed fog servers to cover a country or a continent is a major challenge. For example, the 5G (5th-Generation mobile communication technology) Observatory Biannual Report published in October 2023 shows that the number of 5G base stations in France is 39,502 with 88.8% of population coverage [22]. Each base station can serve more than 2,000 users simultaneously [23]. To achieve a fog service coverage similar to that of 5G base stations, the number of fog servers may need to be even greater than this, requiring major deployment and maintenance efforts. In addition, the population density differs from region to region. Therefore, it may be difficult for deployed fog infrastructure in low-density areas to attract sufficient workloads to produce high resource utilization.

The second issue is the management of a large-scale, public, geo-distributed fog computing platform. The large-scale characteristic means that the platform may contain a large number of fog resources and workloads, which requires a robust and scalable or-

chestration framework to manage them. The public nature of the platform implies that it may need to accommodate many users, which demands effective multi-tenancy strategies to ensure isolation between the users. To this end, we need to simultaneously address the challenges related to scalability and multi-tenancy to manage such a platform efficiently.

The third issue is related to the monitoring of a large-scale geo-distributed fog computing platform. Monitoring plays a critical role in resource usage tracking, failure identification, and workload scheduling, especially in potentially resource-constrained and unstable fog environments. Monitoring a large fog platform is very challenging compared to traditional centralized cloud data centers, as fog infrastructures are deployed in many different locations. An excessive amount of monitoring data transmitted over long distances across a geo-distributed platform may waste the existing network resources and may eventually represent the majority of the system management traffic.

1.1 Contributions

To address these challenges, the thesis proposes two main contributions, which both rely on the concept of cluster federation. A federation is composed of multiple server clusters deployed in different strategic locations that can be managed and used as a single large-scale geo-distributed platform. Our solutions are based on the Kubernetes container orchestrator, the Prometheus monitoring system, and their respective ecosystems, which constitute the current industry standard while remaining open-source, highly mature, and extensible [24], [25]. However, the concepts and algorithms proposed in this thesis can be easily applied and integrated with other current or future container orchestrators and monitoring systems.

The main contributions of this thesis are as follows:

(1) **Multi-Tenancy Management in Scalable Fog Meta-Federations**
Designing large-scale, multi-tenant, public fog federation platforms requires the aggregation of large numbers of clusters in numerous locations covering a country or even a continent. However, it would be difficult for a single organization to deploy enough resources in strategic locations while attracting sufficient workloads to generate high resource utilization. We propose to address this challenge with the vision that many small or medium-sized fog resource providers may choose to cover only a limited region or set of regions. A “global” fog service provider may then expand its service coverage span in additional locations by securing a business deal with the

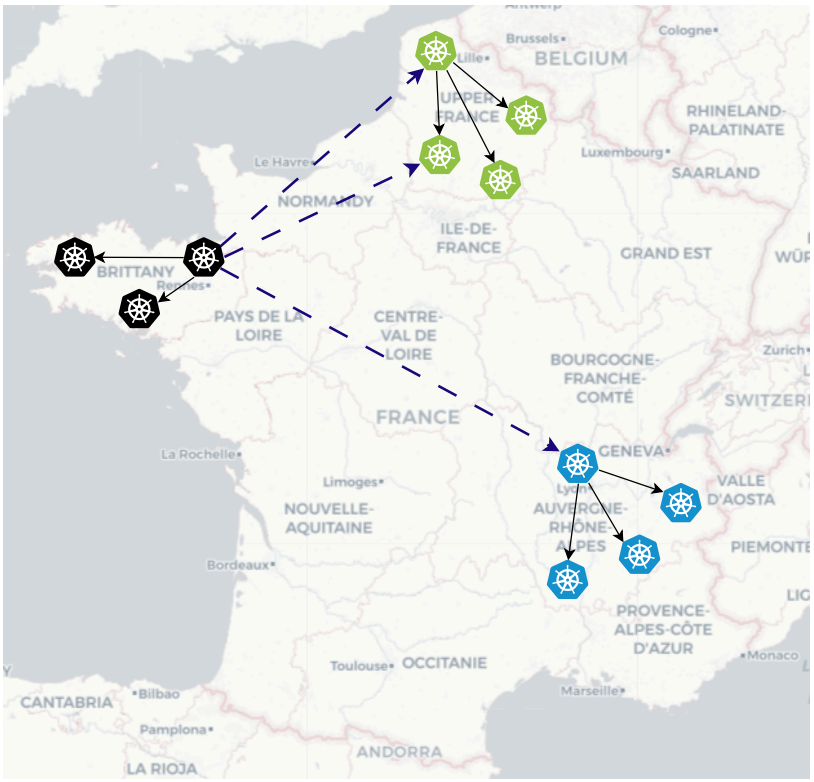


Figure 1.2 – An example of a fog provider in Brittany (black) that wants to expand the service coverage to other regions of France by gaining access to other fog providers in Hauts-de-France (green) and Auvergne Rhône-Alpes (blue). The base image is from OpenStreetMap France [26].

other fog resource providers and by including their resources in a federation. In this way, the global fog service provider may attract new customers and increase the size of its business. At the same time, the regional fog resource providers may choose to lease their resources to multiple fog service providers. For example, as shown in Figure 1.2, a fog provider with its own cluster federation in Brittany may want to use the fog resources of other fog providers in Hauts-de-France and Auvergne-Rhône-Alpes.

This thesis proposes the design of scalable fog *meta-federations*. We define a meta-federation as a complex ecosystem composed of many independent fog resource providers that may set up business agreements with one another to allow access to

their computing resources. The legal framework for setting up such business agreements is outside the scope of this thesis. Any single fog cluster may simultaneously act as the manager of a large federation and contribute its own resources to one or more other federations, which contributes to enhancing resource utilization and reducing the operational cost of the cluster.

Realizing the vision of a large meta-federations ecosystem that can span a country or a continent with thousands of local providers renting their computing resources to hundreds of independent federations in turn requires one to address two main challenges: (i) Multi-tenancy: Workloads submitted by multiple federations to the same member cluster should be strictly isolated from each other, and multiple users from any single federation should also benefit from similar isolation guarantees; (ii) Scalability: Each management cluster [1] must effectively control a large number of member clusters, while each member cluster must be able to lease its resources to a large number of management clusters.

To support the vision of building a large-scale, public, shared, geo-distributed fog computing platform while effectively addressing the complex multi-tenancy and scalability challenges introduced by meta-federations, we present UnBound, a scalable fog meta-federations platform. UnBound relies on Kubernetes to orchestrate resources within individual fog clusters [27] and Open Cluster Management (OCM) to federate multiple member clusters under the authority of a management cluster [28]. We address the issue of multi-tenancy management by isolating federations within a single member cluster using the Virtual Kubernetes Clusters (vCluster) project to create isolated logical sub-clusters within the member clusters [29].

We conduct extensive evaluations through real-world deployments in the Grid'5000 testbed [30] and demonstrate that UnBound achieves inter-user and inter-federation isolation while maintaining comparable application creation time to the original Open Cluster Management and avoiding increasing cross-cluster network traffic between the management and member clusters. Moreover, the resource consumption of UnBound components remains within acceptable limits. Finally, we demonstrate

1. In a cluster federation, a "management cluster" is in charge of deciding which of the "member clusters" will be in charge of handling each newly deployed application.

the stability and scalability of UnBound using federations with up to 500 member clusters and a member cluster belonging to up to 100 independent federations.

(2) **Efficient Monitoring Frameworks in Geo-Distributed Cluster Federations** To enable accurate scheduling decisions, it is necessary to have information about the resource usage status of each member cluster in a geo-distributed Kubernetes cluster federation [31]. This requires a robust monitoring framework that can provide resource utilization data, such as Prometheus and its extension Prometheus Federation [32], [33]. However, the design of Prometheus makes it fetch the precise status of *each available server* with *fixed* frequency. This is unnecessary to allow accurate scheduling, unscalable as the number of servers grows, and it may waste long-distance network bandwidth in a large cluster federation.

In this contribution, we present two frameworks to address these issues in a geo-distributed cluster federation: Acala and AdapPF. Both of them aim to balance between cross-cluster network traffic and the accuracy of monitoring data. Acala exploits two strategies called metrics aggregation and metrics deduplication for reducing the volume of monitoring data that needs to be reported to the management cluster, elevating the traditional view of monitoring from "node" granularity to "cluster" granularity. On the other hand, AdapPF uses a self-adaptive approach to dynamically adjust the scrape interval for each member cluster based on the resource status of the target clusters.

Our contributions based on actual deployments in the geo-distributed Grid'5000 testbed demonstrate that Acala reduces the cross-cluster network traffic by up to 97% and the scrape duration by up to 55% in single member cluster experiments. Our solution also decreases cross-cluster network traffic by 95% and memory resource consumption by 83% in multiple member cluster scenarios. A comparison of scheduling efficiency with and without data aggregation shows that aggregation has minimal effects on the system's scheduling function. On the other hand, AdapPF can achieve comparable application scheduling results to Prometheus Federation with 5 seconds scrape interval while reducing cross-cluster network traffic by 36%.

These two solutions are complementary as they address different aspects of the Prometheus Federation architecture. In principle, they may be combined to leverage the strengths of both.

1.2 Published Papers

The following manuscripts are published as part of this thesis:

Journal article(s)

(1) "Aggregate Monitoring for Geo-Distributed Kubernetes Cluster Federations", **Chih-Kai Huang** and Guillaume Pierre, in IEEE Transactions on Cloud Computing, vol. 12, no. 4, pp. 1449-1462, Oct.-Dec. 2024.

Conference paper(s)

(1) "UnBound: Multi-Tenancy Management in Scalable Fog Meta-Federations", **Chih-Kai Huang** and Guillaume Pierre, in Proceedings of the 17th IEEE/ACM International Conference on Utility and Cloud Computing, Sharjah, United Arab Emirates, Dec 2024.

(2) "AdapPF: Self-Adaptive Scrape Interval for Monitoring in Geo-Distributed Cluster Federations", **Chih-Kai Huang** and Guillaume Pierre, in Proceedings of the 28th IEEE Symposium on Computers and Communications, Tunis, Tunisia, Jul 2023.

(3) "Acala: Aggregate Monitoring for Geo-Distributed Cluster Federations", **Chih-Kai Huang** and Guillaume Pierre, in Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, Tallinn, Estonia, Mar 2023.

1.3 Organization of the Thesis

This thesis is organized into six chapters.

Chapter 2 presents the technical background of this thesis. We first explore the basics of cloud computing with its characteristics, architecture, service models, and limitations. We then discuss geo-distributed fog computing and the reason why it can address some of the limitations of cloud computing. We also introduce virtualization technology and discuss the trend of moving from virtual machines to containers. Finally, we introduce Kubernetes, a widely used open-source container orchestrator that we rely on in this thesis.

Chapter 3 discusses the academic state of the art related to our contributions. We start the chapter with the multi-cluster federation control planes. Then, we review the multi-tenancy frameworks for Kubernetes clusters and position our contributions in this context. Finally, we present the literature on monitoring in fog computing environments and discuss the challenges of the Prometheus monitoring system.

Chapter 4 introduces our first contribution: *Multi-Tenancy Management in Scalable Fog Meta-Federations.* We first define meta-federations as a way to realize the vision of large-scale, public, multi-tenant, geo-distributed fog computing platforms. Next, we present UnBound, a scalable fog meta-federations platform, with its system architecture and multi-tenancy management methods. Finally, we evaluate UnBound in a realistic testbed and show its performance results.

Chapter 5 proposes our second contribution: *Efficient Monitoring Frameworks in Geo-Distributed Cluster Federations.* We first present the Acala monitoring framework and subsequently discuss the AdapPF monitoring framework. Both frameworks follow the discussion with their system design and performance evaluation.

Chapter 6 restates the challenges of building a large-scale, shared, public fog computing platform. Then, we summarize the contributions of this thesis and identify promising directions for future research.

Chapter 2

Background

In this chapter, we first explore the cloud computing concept with its characteristics, architecture, service models and limitations, and then shift to discuss the geo-distributed fog computing. After that, we discuss the evolution of virtualization technology from Virtual Machines (VMs) to containers. We also review Kubernetes and its related topics.

2.1 Cloud Computing

Cloud computing has been one of the most significant technological concepts over the past twenty years [34]. The history of cloud computing can be traced back to 1961 when John McCarthy introduced the first utility computing concept at the Massachusetts Institute of Technology [35]. Afterward, the Compaq Computer Corporation started to bring computing into the business aspect in 1996 [36]. It took almost a decade of evolution for three giants, Amazon, Microsoft, and Google, to roll out their own "cloud" services or platforms successfully. In 2006, Amazon released its cloud computing services, which included Amazon Simple Storage Service (S3) and Amazon Elastic Compute Cloud (EC2) [37], [38]. Then, Microsoft Windows Azure [39] and Google App Engine [40] were also announced and started sharing the cloud market. As of the fourth quarter of 2023, the three major Cloud Service Providers (CSPs) shared a total of around 66% of the cloud market [41].

Cloud computing is changing the way individuals and enterprises develop software and deploy applications. A Eurostat survey of 161,000 EU enterprises with different sizes shows that 45.2% of EU enterprises used cloud computing services in 2023. Compared to 2021, it is 4.2 percentage points greater, representing a growing trend in the business fields [42].

The rising trend of cloud computing can be attributed to the gradual maturation of software. Virtualization technologies such as Virtual Machines (VMs) [43] and containers [44] enable cloud computing as CSPs can leverage them to abstract the computing

resources on a physical server and share each server with multiple users without letting them interfere with each other. Using virtualization technology in cloud platforms brings additional benefits, including improving the overall resource utilization in each server, reducing costs, and providing functions such as high availability and auto-scaling [45]. We will discuss virtualization technologies in Section 2.3.

For users, there are several advantages to using cloud computing such as reducing their investment cost of Information Technology (IT) infrastructures and software. The users can directly utilize the services provided by CSPs without needing to deploy or maintain their own servers and storage systems [46]. Cloud computing can also enhance the flexibility of deployed services such as dynamically scaling the capacity of IT services up and down according to the demand [1]. For instance, a shopping website may experience greater workloads during the weekend, requiring the administrator to scale up the cloud resources to handle the load. After the peak has ended, they can scale resources down to maintain cost-effectiveness. One more advantage that users can benefit from using cloud computing is efficiency. Enterprise users only need to focus on their major business and do not need to care too much about IT-related work. Moreover, applications and data hosted in the cloud can be accessed from almost any device connected to the internet [47].

2.1.1 Cloud Computing Characteristics

Cloud computing relies on numerous servers, networks, and storage resources being placed together in the same location to build a data center that can provide services to cloud users over the Internet. According to the National Institute of Standards and Technology (NIST) definition, the basic characteristics of cloud computing are listed below [48]:

(1) **On-demand self-service:** CSPs provide cloud users with services or computing resources, such as applications, data storage, and infrastructure. It can automatically allocate resources according to user requirements without system administrator intervention.

(2) **Broad network access:** Cloud users are able to access the cloud services with different types of devices, such as mobile phones, laptops, and desktop computers, anytime and anywhere through the Internet.

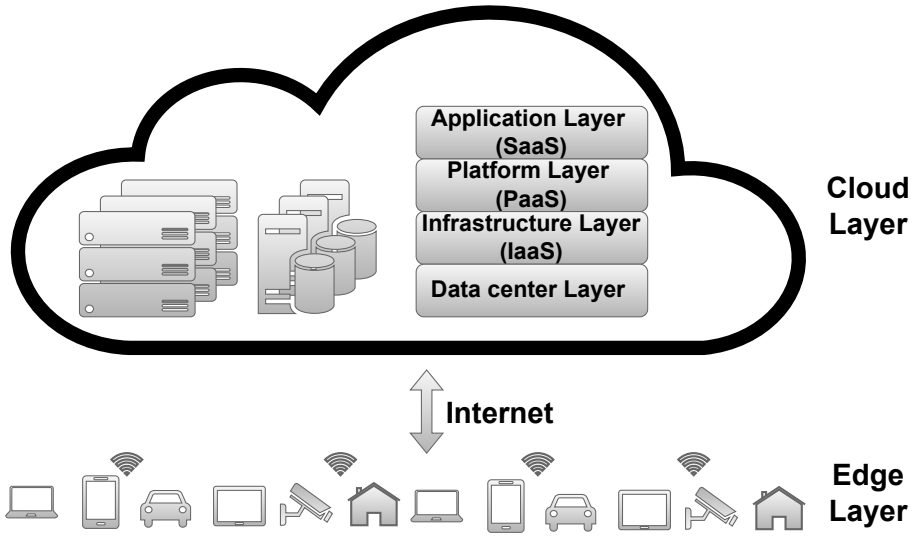


Figure 2.1 – High-level architecture of cloud computing.

(3) **Resource pooling:** The resources managed by CSPs are aggregated into a shared massive computing pool. By using virtualization technologies, CSPs can share the resources or provide services to the end users through a multi-tenancy model.

(4) **Rapid elasticity:** The size of services can quickly scale up and down to adapt to user needs.

(5) **Measured service:** CSPs can monitor the service usage of cloud users and are billed based on the pay-per-use method. Moreover, the CSPs can leverage monitoring data to manage the resources in a data center.

In addition to the above basic characteristics, there are some common characteristics, including massive scale, resilient computing, and geographic distribution service orientation [49]. To sum up these characteristics, cloud computing is the integration and development of distributed computing, Internet technology, and large-scale resource management.

2.1.2 Cloud Computing Architecture and Business Models

Figure 2.1 presents the relationship between cloud computing and its end users. The cloud layer aggregates computing resources into one or more data centers and provides services to the end users. The edge layer is composed of end users, including their Internet-of-Things (IoT) devices, mobile phones, and computers. These devices produce the data

and requests, and send them through the Internet to the cloud data centers for analysis or processing.

The architecture of a data center could be divided into four layers, which are application, platform, infrastructure, and data center layers [46], [50]. Moreover, NIST classifies the service models into three levels: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [48]. SaaS is oriented towards end users who only use complete applications. It provides Internet-based on-demand software services without requiring users to install or maintain them. PaaS targets application developers, in which PaaS delivers a platform that contains software development and management frameworks. Developers only need to upload code and data to use the platform without having to worry about the issues related to the underlying network, storage, and operating system. IaaS is designed for users who require complete control over their computing infrastructure, including servers for computation, storage, and networking. Each service model can be mapped to the different architecture layers, and we discuss the architecture from the down to the top layers of cloud computing.

(1) **Data center layer:** CSPs operate the data centers to offer the services to the cloud users. Each data center includes thousands or more physical machines, such as servers, switches, and routers, packed into racks and connected by a high-bandwidth internal network. There are several challenges in this layer, including the configuration of hardware components, ensuring fault tolerance, and keeping energy efficiency.

(2) **Infrastructure layer:** This layer, also called the virtualization layer, leverages virtualization technologies to share computing resources with different cloud users. The infrastructure layer belongs to the IaaS model, providing computing resources, such as servers, storage, and networking. Some well-known IaaS products are Amazon Elastic Compute Cloud (Amazon EC2) [51], Google Cloud Storage [52], and Azure Virtual Network [53].

(3) **Platform layer:** The platform layer stands in the PaaS model and is built using the infrastructure layer. It is responsible for providing application frameworks to the software developers. It can relieve developers from the burden of managing servers and deployment settings. Google App Engine [54] is one of the products in this layer.

(4) **Application layer:** This layer can map to the SaaS model and aims to offer different cloud software services to the cloud users, such as E-mail services and document

editor. Typical examples are Google Workspace [55], which include Google Docs and Gmail.

2.1.3 Cloud Computing Deployment Models

Cloud users can choose between different deployment models for their applications based on their requirements, such as locations and policies. Cloud data centers, managed by CSPs such as Microsoft Azure, Google Cloud, and Amazon Web Services, provide the services mentioned in the previous section in the form of *public cloud*, where cloud users do not need to maintain their own infrastructure and software. For cloud users who want a higher degree of performance, reliability, and security, an alternative is to select a model named *private cloud*, where cloud users build their own data centers to fulfill the needs of control or privacy.

Some users need their applications to run in specific geographic locations for performance or legal reasons, while others wish to avoid a single vendor lock-in. For them, *multi-cloud* is a viable solution for geo-distributed application deployment [6], [56]. The administrators of applications can launch multiple applications by using different public cloud service providers in various locations to serve end users. For example, Google Cloud and Microsoft Azure operate data centers in Taiwan. However, AWS does not have one. If AWS customers want to deploy an application to serve end users closer to Taiwan, they need to leverage other cloud providers, ending up with a multi-cloud deployment. Another geo-distributed deployment model is the *hybrid cloud*. The idea of hybrid cloud deployment is to combine the resources from one or more private data centers with the public cloud, which brings benefits from both sides. The users can keep sensitive workloads or data in a private data center and utilize the scalability and flexibility of the public cloud to run a larger number of applications and less sensitive workloads.

2.1.4 Cloud Computing Limitations

Cloud computing brings many benefits, but it also presents challenges in different aspects. The first major issue for cloud computing is its energy consumption. A report from the International Energy Agency (IEA) shows that the electricity use for cloud data centers and transmission networks each is estimated up to 1.5% of the global use [57]. Moreover, the growing trend of data centers is driving an increase in energy usage by Artificial Intelligence (AI) [58]. Security and privacy are also significant topics that people

care about. According to a survey from Cloud Security Alliance (CSA), the top threats are data breaches, weak identity, credential and access management, and insecure Application Programming Interfaces (APIs) [59]. Another report shows that some of the threats are growing over the years such as data breaches [60].

The enhancement of network accessibility and bandwidth, combined with the widespread proliferation of cloud data centers in different locations worldwide, has significantly reduced end user to cloud service latency. As a result, popular cloud services such as Facebook can be accessed within as little as 40 milliseconds round-trip latency [61]. The latency between end users and cloud services is an important topic for CSPs because lower latency brings a better user experience and thereby it impacts the profit of application owners. Amazon discovered that every extra 100 milliseconds of delay resulted in a 1% loss in sales [62]. Meanwhile, another study also demonstrated a similar outcome that an increase of 0.5 seconds in generating search results causes a 20% decrease in traffic [63]. Reducing network delays also enables the development of new latency-sensitive applications. For example, virtual reality and 360-degree video streaming require the total end-to-end latency, which includes both network transmission and application processing delays, to remain within 20 milliseconds [8].

The rapid development of the Internet of Things (IoT) enables the creation of smart home and improved urban services with the smart city. A forecast shows that the volume of data generated by IoT devices will reach 79.4 zettabytes by 2025 [64]. Meanwhile, to monitor and analyze the data collected from various IoT devices and sensors, the data must be sent to a cloud data center for real-time data processing. Long-distance data transmission of such large amounts of data over the network may eventually saturate the existing network links [12].

Although cloud users can take advantage of multi-cloud deployments to execute their applications in different data centers with multiple CSPs to reduce the latency and the volume of long-distance data transmission over the network, the centralization of large data centers means that they may remain physically distant from the end users [4]. As these limitations of cloud computing become more recognized, fog computing emerges as a solution to address their limitations.

Table 2.1 – Differences between fog computing and cloud computing.

Characteristics	Fog Computing	Cloud Computing
Architecture	Decentralized fog nodes or clusters	Centralized data centers
Latency Between Users and Nearest Servers	Low	High
Distance From Users	Close	Far
Bandwidth	Low	High
Computing Capacity	Intermediate	High
Storage Capacity	Intermediate	High
Use Cases	Latency sensitive or IoT applications	General applications

2.2 Fog Computing

Fog computing was proposed by Cisco in 2012 [13] as a widely distributed cloud-liked infrastructure to address the limitations of centralized cloud computing. The aim was to bridge the gap between end users/IoT devices and traditional cloud computing data centers by providing resources, including computing, storage, and networking services, that are closer to them. This design can fulfill the characteristics of IoT applications, such as geographic distribution and low latency.

In 2017, the OpenFog Consortium Architecture Working Group published a white paper called "OpenFog Reference Architecture for Fog Computing" [65] to further consolidate the definition of fog computing. Then, in 2018, the IEEE adopted this standard [66]. In this standard, the authors state that "*Fog computing is a horizontal, system-level architecture that distributes computing, storage, control, and networking functions closer to the users along a cloud-to-thing continuum.*" Based on this definition, fog computing is seen as an extension of cloud computing where the computing resources are spread in different locations close to the data producers and users within a large geographical coverage. Fog computing should have all the characteristics of cloud computing, such as virtualization, service models, and efficiency. To conclude these two definitions from Cisco and OpenFog, they share a similar concept: computing resources are provided between end users and cloud data centers to reduce the end-to-end latency of applications. We compare the main differences between fog computing and cloud computing in Table 2.1.

The main ideas of fog computing and edge computing are similar in that both of them address the latency issues between end users and cloud data centers. However, there are still two key differences between these two concepts. First, fog computing includes cloud

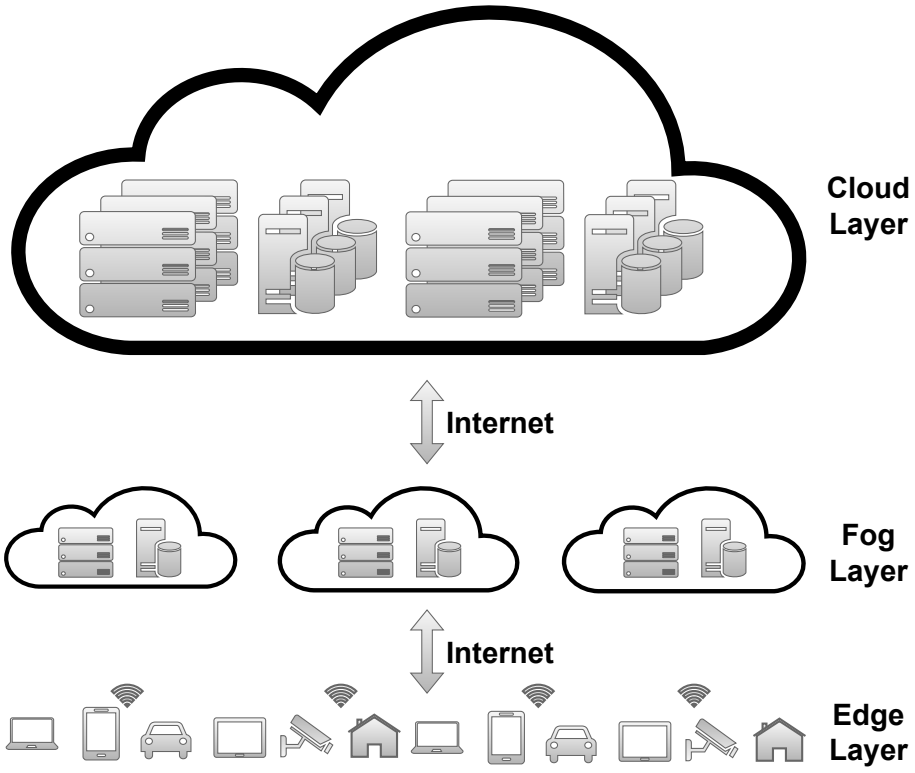


Figure 2.2 – High-level architecture of fog computing.

computing to define a complete computing continuum, whereas edge computing excludes cloud computing as an independent architecture. Second, the structure of fog computing is hierarchical, combined with cloud, fog, and edge devices, while edge computing usually involves a flat structure with fewer layers [65].

2.2.1 Fog Computing Architecture

Figure 2.2 illustrates a typical high-level fog computing architecture, which includes edge, fog, and cloud layers.

(1) **Edge layer:** This layer, also called the device layer, is the bottom layer in fog computing. The edge layer contains different types of devices such as IoT devices, sensors, mobile phones, smart vehicles, and other endpoints that can connect to the Internet. These devices generate or collect data, and send them to the upper layer for additional processing, such as analysis and decision-making. To send data, the methods of accessing the network are often wireless using protocols, such as Wi-Fi,

cellular network (LTE, 5G), and Long Range Wide Area Network (LoRaWAN) [18], [67].

(2) **Fog layer:** This layer is composed of computing resources located near the data sources outside traditional cloud data centers. These resources may potentially cover a very large region, such as a country and a continent. Moreover, fog servers often have limited computational power. For example, Raspberry Pi single-board computers are often used to build fog clusters [68]–[71]. Additional devices can also join the system and provide computational power, such as drones [72] and vehicles [73], which can also serve as part of the fog infrastructures. Applications that want to execute close to their end users will deploy in this layer.

(3) **Cloud layer:** This layer is made up of one or more powerful cloud data centers. Each data center consists of high-performance servers, high-speed network connections, and high-capacity storage. These computing resources can be used for applications that require reliability and high performance. Since these cloud data centers are physically far from end users, this layer can deploy the non-latency-sensitive part of fog applications.

2.2.2 Fog Computing Applications

The emergence of fog computing paradigms presents new opportunities to serve end users in close proximity and process data from sources outside traditional on-premise cloud data centers. The main idea of this design is to improve the user's Quality of Experience (QoE), especially for those applications that are not compatible with traditional cloud computing deployment [7].

Latency-sensitive applications require low end-to-end latency between users and applications. For example, humans have a low tolerance for delays or inconsistencies. Therefore, applications such as virtual reality and 360-degree video streaming need an end-to-end round-trip latency under 20 milliseconds [8]. Using a distributed fog infrastructure can reduce latency to meet the needs of latency-sensitive applications.

Fog computing resources located close to end users can also bring benefits to applications such as video surveillance. These applications produce large volumes of data and require broad network bandwidth to transmit these data to cloud data centers for processing. By using fog computing, processing can take place in closer infrastructures, which effectively minimizes the volume of data transmitted to the cloud.

Although web applications usually do not require ultra-low latency between end users and applications, as mentioned above, excessive latency for end users may reduce not only the profit but also the traffic. Improving the user experience is one of the main goals of fog computing. These fog infrastructures can be used for web content delivery and caching, such as static items (web pages, images, and videos), as well as application services [74], [75], to reduce end-to-end latency.

Several works have applied this distributed computing paradigm to different fields, including the fields of transportation [73], [76], smart city [77], [78], agriculture [79], [80], and entertainment [81].

2.2.3 Fog Computing Challenges

Fog computing addresses the limitations of cloud computing, such as high latency and long-distance network transmission. However, to fully realize the fog computing potential, there remain challenges that need to be tackled. We discuss each point as follows:

(1) **Computing resource constraints:** Traditional cloud data centers are composed of many powerful servers, massive volumes of storage, and stable networks. Instead, fog infrastructures are equipped with potentially weak servers, small storage, and unstable networks. This challenge requires methods to handle workloads efficiently within these limits, making sure that users can still have a similar user experience to cloud computing. We discuss this constraint further in Section 2.3.

(2) **Scalability challenges:** Fog computing widely distributes fog infrastructures to strategic locations near the end users and data sources. These infrastructures may cover a very large region, such as a city, a country and even a continent, and may therefore be composed of a very large number of computing nodes. Maintaining a scalable fog platform demands a robust framework with a strong orchestrator to govern the fog resources as well as handle many functions, such as deployment, scheduling, and monitoring, which are utilized by different users. A fog platform may need to handle a large number of users or administrators. Therefore, it is crucial to deal with the multi-tenancy challenge that users may have from different departments or even organizations. Additionally, the network traffic for management with a large number of infrastructures in the platform is also an issue that needs to be taken into account. We discuss this challenge in Section 2.4.

(3) **Security and privacy issues:** Cloud data centers implement different security measures to protect cloud users. Their centralized design makes it easier for CSPs to build their security ecosystem, which includes physical and network security since all the computing resources are in the same location. In centralized data centers, physical infrastructures can be secured by surveillance cameras and security guards, and the network can be protected by specifically designed machines, such as network firewalls and Intrusion Prevention Systems (IPSs) [82], [83]. In contrast, fog computing is a geographically distributed architecture, which makes security and privacy issues more challenging. Distributed fog infrastructures may be vulnerable to physical tampering or theft. Network protection in fog computing may require the use of distributed firewalls or IPSs rather than a single machine. To ensure data privacy, a centralized data center makes it easier to precisely locate data and computation. However, distributed fog resources within a single fog platform may be located in different countries with different data compliance regulations.

This thesis aims to address the second point mentioned above, which is the scalability challenge in geo-distributed fog computing. Scalability is a key concern in this environment that enables the fog platform to efficiently handle increasing infrastructures, workloads, users, and management traffic. As a result, designing frameworks to enable the evolution of scalable fog computing platforms is the main objective of this thesis.

2.3 Virtualization Technology: Virtual Machines and Containers

Virtualization is a key technology that enables multiple users to share computing resources in a physical server without interfering with each other. By doing this, virtualization improves computing resource utilization and facilitates functions such as on-demand scaling [84]. There are two main types of virtualization for computing: hardware virtualization and operating system virtualization [2]. While numerous virtualization technologies also exist for network or storage, they fall outside the scope of this thesis and will not be discussed here.

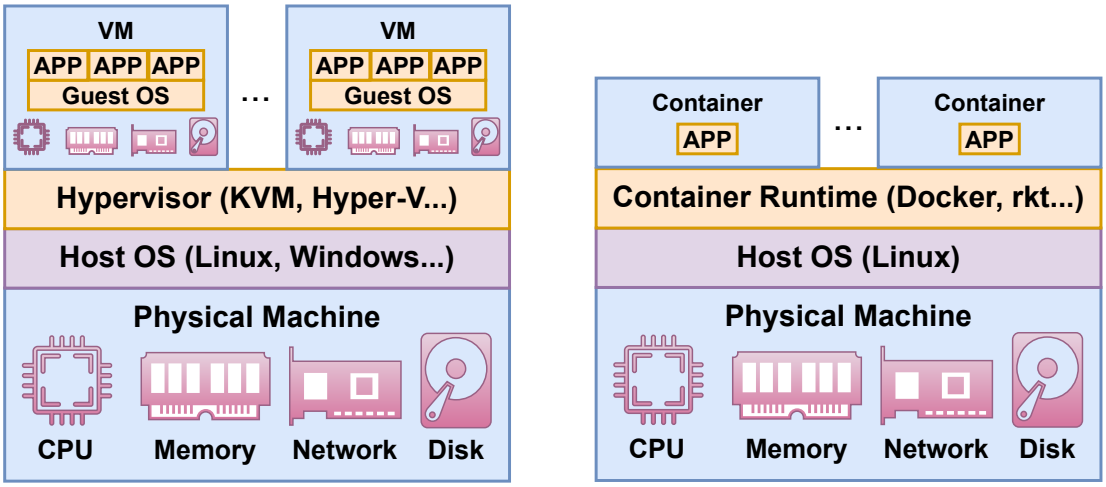
Hardware virtualization, also called hypervisor-based virtualization, is a technology to run multiple Virtual Machines (VMs) in a single physical machine [2], [85]. As shown in Figure 2.3(a), each of the VMs has its own virtual computing resources, operating

system, and applications, which provide the same user experience as a physical server. For example, users can install any applications in a VM or use any kind of operating system. To manage the computing resources of VMs, a hypervisor or Virtual Machine Monitor (VMM) is required. The hypervisor allocates resources to each VM and manages the scheduling between VM resources and the physical hardware. Although a VM has its own virtual computing resources, the execution of computing tasks is carried out by the physical hardware. Well-known hypervisors include Microsoft Hyper-V [1], VMware ESXi [2], Xen [3], and VirtualBox [4].

Another type of virtualization is Operating System (OS) virtualization, also known as containerization. It is an approach that encapsulates an application, along with its essential libraries, dependencies, and execution environment, within a container image [2], [85]. Containers present the characteristics of portability and isolation. By using the same image, applications can have a consistent execution in different computing environments. Different from VMs, containers rely on a shared image registry such as the Docker Registry [86], which allows users to easily download application images and run them in any environment equipped with the container runtime. Container isolation in the same execution environment is provided through the use of the Linux kernel features, such as namespaces and control groups (cgroups) [87]–[89]. Well-known container technologies include Docker [5], LXC [6], and Podman [7].

Although VMs are the backbone of centralized cloud computing architecture, the computing resources of fog infrastructures are often limited and geographically distributed compared to cloud data centers. Using VMs to deploy applications in this type of machine is very challenging. VM-based applications include applications related to software, dependencies, and data and contain an entire guest OS, which potentially consumes significant amounts of computing resources. This situation leads to high overhead for VMs, which might not be acceptable on constrained fog devices. Moreover, this overhead makes it hard to scale the number of VM-based applications per server. VM image sizes are usually large also due to the guest OS, often reaching into the gigabytes [74], [90]. Using VMs in this context may also cause an increase in the launch time of applications to

1. Microsoft Hyper-V - https://reurl.cc/dLYLAg
2. VMware ESXi - https://reurl.cc/1393qV
3. Xen - https://reurl.cc/g4d4EN
4. VirtualBox - https://reurl.cc/orjrAj
5. Docker - https://reurl.cc/eLrlXx
6. LXC - https://reurl.cc/RWNGvr
7. Podman - https://reurl.cc/VN9O8R



(a) Hardware virtualization

(b) OS virtualization

Figure 2.3 – Comparison of hardware virtualization (a) and OS virtualization (b) architectures.

serve users, which creates significant delays in fog computing, whereas the goal of fog is to provide a better user experience.

To effectively address the challenges related to performance and scalability encountered with VMs in fog computing environments, OS virtualization offers a more suitable solution. The main reason is the lightweight nature of containers, which do not need to contain a full guest OS, as illustrated in Figure 2.3(b). This characteristic ensures low consumption of computing resources, which is especially important in constrained fog infrastructures. Furthermore, the image sizes can be greatly reduced as they do not include a full OS. This reduction of the size allows containers to have faster startup time than VMs. Typically, it takes a matter of seconds or less for a container to start and turn ready status to handle user requests [74], [88]. Additionally, multiple solutions exist to further reduce the launch time of a container [69], [91]–[93]. As a result, container-based applications have a fast launch time and are able to scale the number of applications to a higher degree across geo-distributed fog devices, which is crucial for adapting to the dynamic demands in fog computing scenarios. Many works use containers to enhance the resource utilization efficiency and reduce service latency within fog computing environments [18], [94]–[97].

Because each physical machine may run a large number of containers, fog infrastructures are requested to manage thousands or even tens of thousands of containers. Managing these amounts of containers spread across numerous machines in a fog computing

environment requires a robust orchestrator to handle deployment, scaling, and networking seamlessly and efficiently. Various container orchestrators have been proposed, such as Docker Swarm [98], Apache Mesos [99], and Kubernetes [27]. Among these orchestrators, Kubernetes has now become the most widely-used platform [100]–[102].

On the other hand, there is still no standardized platform for fog computing that can support all its specific requirements, especially regarding scalability. Therefore, this thesis considers containerization as a viable virtualization solution for geo-distributed fog computing environments. Owing to the fact that Kubernetes is currently the de-facto standard for cloud scenarios, we choose it as the preferred orchestrator for managing containers and clusters, aiming to explore ways to meet the scalability requirements of fog computing. Kubernetes has been adopted in many academic works for this type of projects [18], [71], [103]–[105].

2.4 Kubernetes

Kubernetes is an open-source container orchestrator, often abbreviated K8s, which was initially designed by Google. Later, Kubernetes was donated to the Cloud Native Computing Foundation (CNCF). In 2018, CNCF accepted Kubernetes at the "graduated" maturity level, which certifies that Kubernetes is a stable and production-ready platform [24]. As an open-source project, it has attracted around 3,600 contributors and is very active in releasing new versions [106]. This level of activity can speed up bug fixes and feature deployment to accommodate the rapidly evolving needs of its users and can keep Kubernetes at the forefront of container orchestration technologies.

Kubernetes can be used to automate the deployment, scaling, and management of containerized applications in public or private cloud infrastructures. It is deployed on a set of computing nodes that constitute a cluster. Each Kubernetes cluster is composed of two roles: control plane and worker node. The control plane is in charge of managing worker nodes and containers. To provide fault tolerance and high availability in a production environment, the platform administrator can also install multiple control planes that are distributed across several machines. Moreover, each cluster also needs at least one worker node to run the containers. Kubernetes relies on container runtimes to execute

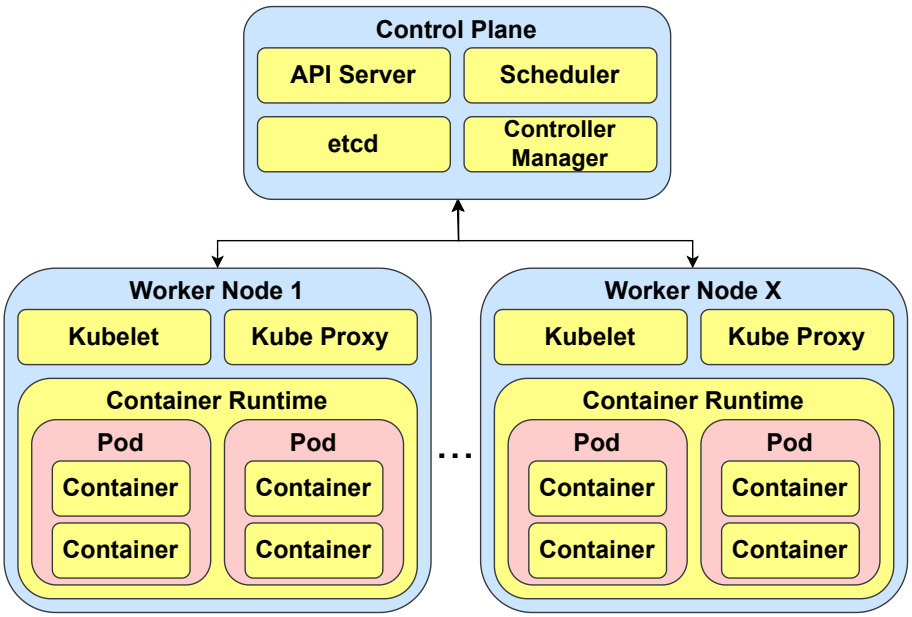


Figure 2.4 – Simplified Kubernetes architecture.

the containers, such as containerd [8], CRI-O [9], Docker Engine [10], and Mirantis Container Runtime [11] on all nodes in a cluster.

2.4.1 Architecture

Figure 2.4 presents a simplified view of the Kubernetes architecture. The control plane includes four main components: API server, scheduler, etcd, and controller manager.

(1) **API server:** API server exposes the Kubernetes API to the system administrators and other components. The Kubernetes API provides a standard interface to interact with the Kubernetes platform and perform tasks such as deploying and managing applications. In the Kubernetes API, a resource is an endpoint that stores API objects. For example, the built-in *Pod* resource contains a collection of Pod objects (we discuss the concept of *Pod* in Section 2.4.1.1). Moreover, it is possible to launch multiple API servers and enable load balancing to distribute requests across these API servers in a Kubernetes cluster.

8. containerd - https://reurl.cc/orWqkg
9. CRI-O - https://reurl.cc/bD0yQE
10. Docker Engine - https://reurl.cc/G4Wl8A
11. Mirantis Container Runtime - https://reurl.cc/M4qGp3

(2) **Scheduler:** Scheduler is responsible for monitoring newly initiated Pods without an assigned worker node and determining an appropriate worker node for their execution. A Pod can be scheduled using different indicators, such as affinity/anti-affinity specifications, policy constraints, and hardware requirements.

(3) **etcd:** etcd is a key-value data store with features such as strong consistency, distributed design, and high efficiency. The task for etcd is to store all the objects in a Kubernetes cluster. In a production environment, etcd is usually deployed in an etcd cluster with an odd number of servers such as 3 and 5 to ensure high availability. This is because etcd is based on the Raft [107] algorithm to keep data consistent across multiple etcd, which requires a majority of members to accept updates to the etcd cluster status. Similar to Kubernetes, etcd is also a CNCF graduate project [108].

(4) **Controller manager:** Controller manager oversees the different controller processes in a Kubernetes cluster. It comprises a series of controllers, such as node controller, job controller, and deployment controller. Each controller monitors the status of their target Kubernetes resources through the API server and ensures that the actual resource status converges toward the desired status.

The second role in Kubernetes is the worker node, which is responsible for running the containerized applications placed by the scheduler in the control plane. Each worker node executes three main elements: Kubelet, Kube-proxy, and container runtime.

(1) **Kubelet:** Kubelet is an agent in charge of managing the lifecycle of containers within a worker node. Kubelet continually *pulls* the latest information about the desired status from the API server and ensures that the status of corresponding containers in the worker node matches their *PodSpec*. A *PodSpec* is a definition of the intended behavior of the Pods. In addition to pulling information, Kubelet also periodically reports the status of the worker node and Pods to the API server.

(2) **Kube-proxy:** Kube-proxy is tasked with setting up network communication for Pods in the Kubernetes cluster. It runs in each worker node and manages the network rules to allow Pods to communicate with each other internally within the cluster and with the external world.

(3) **Container runtime:** Container runtime is the component that allows Kubernetes to actually run containers. As motioned above, there are several runtimes for Kubernetes so that the platform administrator can choose the preferred one to deploy in their cluster.

Several add-ons can also be deployed in a Kubernetes cluster to provide cluster-level functionalities, including Domain Name System (DNS) services, network plugins, and monitoring systems. In this thesis, one of the contributions focuses on the scalability of the monitoring system, especially for the Prometheus monitoring system. This topic is discussed in Section 2.4.5. The platform administrator can select from various CNCF projects to fulfill specific needs. These projects can be installed easily through package managers such as Helm [12].

2.4.1.1 Workloads

In the Kubernetes ecosystem, the smallest execution unit is a *Pod*. A Pod is defined as a group of one or more containers that are scheduled on the same worker node and managed together by Kubernetes. The most frequent use case uses a single container per Pod [109]. In the case of containers that need to work together intensively, the users can run multiple containers in a single Pod. This is because containers in a Pod share computing resources, dependencies and volumes, and can communicate with each other through a local host using different port numbers.

To enable both high availability and application scalability, Kubernetes offers different types of built-in resources for managing Pod replication efficiently, including *Deployment*, *StatefulSet*, and *DaemonSet*.

Deployment resource is designed to manage stateless applications, such as web servers and API backends, that do not need to store data generated by end users. The administrators first describe the desired state of Pods in a YAML or JSON file. The Deployment controller then watches the running application state and continually ensures that the status between the desired and the observed state is consistent. For example, the administrators can change the desired number of Pods anytime. The Deployment controller will automatically handle the related operations, such as creating and deleting Pods. This resource can be used to quickly scale the number of replicas up or down or automatically replace crashed Pods.

StatefulSet resource is utilized to handle stateful applications such as databases and data stream processing systems that require a stable identity or persistent storage for storing data. StatefulSets are similar to Deployment resources in that the administrators provide a YAML or JSON file with desired specifications. Different from a Deployment resource, Pods in a StatefulSet can be deployed and scaled in a strict sequential order.

12. Helm - https://reurl.cc/dLRKek

This characteristic is particularly crucial for micro-services that have a specific processing order.

DaemonSet resource is typically used when administrators want to deploy applications to all nodes. The DaemonSet ensures that all nodes in a Kubernetes cluster will run the application, which is for instance suited for log collection and monitoring applications.

In addition to the standard built-in resources, Custom Resource Definitions (CRDs) can be used to create new types of resources, as we discuss next.

2.4.1.2 Custom Resource Definitions (CRDs) and custom Kubernetes controllers

Custom Resource Definitions (CRDs) are a powerful mechanism for extending the Kubernetes API. It allows administrators to create and manage new Custom Resources (CRs) beyond the default built-in resources. When new CRDs are deployed in a Kubernetes cluster, the API server creates new RESTful paths and handles the whole lifecycle for these CRs. Users can interact with them in the same way as with built-in resources. CRs are increasingly being used to implement core functionalities within the Kubernetes framework [110]. Moreover, this concept has been used in several Kubernetes-related projects. For example, ManifestWork is a CR used to manage Kubernetes resources across multiple Kubernetes clusters in the Open Cluster Management open-source project [111].

The controller pattern in Kubernetes is used to run a control loop, which repeatedly tracks the current status of objects [112]. A controller continually makes sure its actual state is the same as the desired status specified by the user. Without the custom controller designed to execute actual logic for a CR, the CR would only store objects in the Kubernetes cluster and only be used to store and retrieve structured data. It is therefore necessary to run a custom controller to manage the CR and continuously synchronize and update its status.

Custom controllers, also called operators, can be designed following the control pattern by using the Monitor, Analyze, Plan, and Execute over a shared Knowledge (MAPE-K) principle or any domain-specific logic [113], [114]. This can be done by frameworks in different coding language, such as Kubernetes Operator Pythonic Framework (Kopf) [13], Java Operator SDK [14], and Kubebuilder [15].

13. Kopf - https://reurl.cc/kr4aNx
14. Java Operator SDK - https://reurl.cc/YVM0Wn
15. Kubebuilder - https://reurl.cc/OG2jpr

2.4.1.3 Pull versus Push Management Model

In this thesis, the definition of the push management model is that the control plane watches resource APIs and pushes the resource manifests to the worker nodes. This push action is done by the control plane directly accessing each worker node and managing all workloads, which is a simple method to manage resources in a cluster. It has some advantages such as faster propagation of changes across the cluster. However, the push model may have scalability challenges because the central controller manages all the resources in a cluster and thereby becomes a bottleneck.

In contrast to the push model, the pull model means that an agent deployed in each worker node periodically monitors the resource APIs defined in the control plane, fetches the resource manifests and applies them to its corresponding worker node. By offloading management tasks to agents distributed on each worker node, the control plane can reduce management pressure, which improves system performance and then increases scalability. As a result, the pull model is considered more robust and scalable.

2.4.2 Scalability

A single Kubernetes cluster contains many worker nodes with many Pods each. It would be very challenging for the control plane to directly operate all the resources in a large cluster [115]. To address this issue, Kubernetes chose the *Pull* model to manage the cluster. Kubelet pulls the desired state of the Pods from the API server and makes sure that the corresponding Pods have the same status as desired.

However, a single Kubernetes cluster still has size limitations to keep performance and stability. The Kubernetes documentation suggests that the size of a single Kubernetes cluster should not exceed a specified limit: each worker node should not run more than 110 Pods, and the whole cluster should not exceed 150,000 Pods. Moreover, the number of worker nodes should remain under 5,000 nodes [116].

These scalability limitations may not allow a single-cluster platform with the size of a very large geo-distributed fog computing platform. To overcome issues related to the scalability of a single Kubernetes cluster, deploying multiple clusters has emerged as a viable solution. The platform can then be scaled by launching several clusters and managing them together. However, managing these clusters efficiently is a difficult challenge. As a result, it quickly becomes desirable to organize the multi-cluster platform as a "federation" of multiple independent clusters, each of which is in charge of its own resources and

components. By doing so, the administrators are able to manage the resources of multiple independent clusters as a single homogeneous cluster.

2.4.3 Federations

The emergence of the Multicluster Special Interest Group (SIG) [117] from the Kubernetes community aims to solve the issues of multi-cluster administration and application management in multiple Kubernetes cluster environments. The SIG proposes different APIs to deal with the challenges in this environment. For instance, the goal of the About API [118] is to enable the identification of clusters within a *ClusterSet* (a group of clusters), and the purpose of the Work API [119] is to deploy workloads across different clusters within a *ClusterSet*.

In addition to the above APIs, the Multicluster SIG presents a federation solution called Kubernetes Cluster Federation (KubeFed), which provides application deployment and resource management in multiple Kubernetes cluster environments [120]. The KubeFed platform is typically organized into one management cluster and multiple member clusters. The management cluster determines which member clusters will handle each newly deployed Pod. Users can manage multiple Kubernetes clusters from a single host cluster with the KubeFed control plane installed.

KubeFed extends Kubernetes with CRDs to offer abstractions such as *FederatedNamespaces* and *FederatedDeployments* for managing multi-cluster federated resources. It also introduces three concepts for these resources: *Template*, *Placement*, and *Override*. *Template* specifies the desired state of the federated resources across all member clusters. *Placement* defines the member cluster where federated resources should be deployed. If this field is empty, KubeFed will not be distributed to any cluster. *Override* allows the user to customize the configuration for specific member clusters. For example, it can be used to change the number of replicas for a particular cluster. Based on these abstractions and concepts, users can deploy their applications to different Kubernetes clusters with a number of Pods and where they should run that under KubeFed control. We discuss KubeFed in detail as well as other federation frameworks in Section 3.1.

2.4.4 Multi-Tenancy

By default, Kubernetes is designed for environments where all users trust each other. However, similar to sharing a server with virtualization technology, a Kubernetes cluster

may need to support many different users to deploy their applications and services thereby saving costs and simplifying administration.

Supporting multi-tenancy and isolating the workloads of multiple tenants can be realized by making each user deploy applications in a separate Namespace and using Role-Based Access Controls (RBAC) to restrict each user in their Namespace and to scope security policies to specific Namespaces [121]. This method is considered a "soft" form of isolation as all tenants share the same control plane, and appropriate configurations are required to isolate their data planes.

On the other hand, "hard" tenant isolation is more difficult to achieve. One possibility is to create a separate Kubernetes cluster for each tenant so that both the control and data planes are totally separated from each other. However, the cost of launching multiple clusters is high, and it may be hard to manage these clusters.

Another method for hard isolation is using virtual control planes in a single Kubernetes, where each tenant has their own control plane. For example, each tenant may store metadata in separate databases, which prevents data leakage between tenants. In terms of data plane isolation in this context, there are two different designs: (i) Tenants utilize a shared data plane to enhance resource utilization while isolating Pods through Kubernetes Namespaces. (ii) Each virtual control plane is assigned its own worker nodes, which provide stronger isolation for user applications.

The isolation can be only for the data plane by reserving specific servers within a Kubernetes cluster for tenants. Each tenant uses the same control plane and deploys the applications to their own worker nodes. We discuss multi-tenancy frameworks in the Kubernetes environments further in Section 3.1.3.

2.4.5 Monitoring

Monitoring is an essential functionality for modern computing systems to keep the system healthy and improve its resource utilization. The demand for monitoring becomes greater with the increasing complexity of systems, which requires monitoring of large numbers of entities ranging from bare metal machines to software objects, such as VMs and containers.

Kubernetes is a complex system that is composed of many components, such as worker nodes, Kubernetes resources, networking, and controllers. Monitoring these objects not only allows real-time understanding of their status but also traces the history data for debugging or risk prediction. Moreover, accurate monitoring data is necessary to efficiently

schedule applications on a set of available resources. However, monitoring a great number of components is very challenging. For example, monitoring may produce a huge amount of data that needs to be stored in the cluster, which requires sufficient storage space. In addition, this amount of data may waste the network bandwidth to transfer them within the Kubernetes cluster.

2.4.5.1 Prometheus

Prometheus is an open-source monitoring and alerting software. Similar to the Kubernetes project, it is also a graduate project from CNCF, which shows Prometheus is stable for production and has great potential to integrate with modern orchestrators such as Kubernetes [25].

The Prometheus ecosystem consists of three main components: Prometheus server, exporters, and alertmanager. The Prometheus server is responsible for scraping monitoring data, and storing them in a time-series database. The term "scrape" represents the action by Prometheus of fetching metrics from targets. The administrator can set a scrape interval to periodically pull the monitoring data. The default scrape interval is 60 seconds, which means that the Prometheus server scrapes the metrics every 60 seconds. The administrator can query these stored metrics using the Prometheus Query Language (PromQL). PromQL is designed to apply mathematical operations and data aggregation functions to time-series data. Prometheus can also be integrated with visualization tools such as Grafana [16].

Prometheus uses HTTP to pull metrics values from remote targets. For monitoring applications or services that do not have native Prometheus metrics endpoint, exporters can be used for converting metrics from target systems into a format that Prometheus can pull. Prometheus officially maintains several exporters such as Node-exporter [122].

Alarms are another important part of a good monitoring system. In Prometheus, the tasks of scraping data and issuing alarms are separated into two components. The administrator can define alerting rules in the Prometheus server and let the server periodically evaluate the rules. When the alerting conditions are met, the Prometheus server will push alerts to the alertmanager. The alertmanager handles these alerts and sends the alarm messages to the users.

In addition to deploying Prometheus in a single Kubernetes cluster, Prometheus also provides a function called Federation. As shown in Figure 2.5, this feature allows a

16. Grafana - https://reurl.cc/eL0Y1L

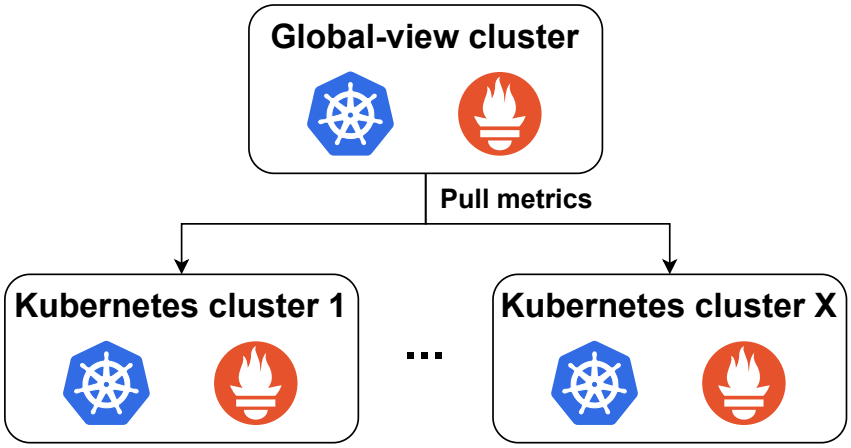


Figure 2.5 – Architecture of Prometheus Federation.

Prometheus server to gather monitoring data from other Prometheus servers, which can therefore build a global-view cluster and scale up to monitor multiple Kubernetes clusters. By querying the monitoring metrics from the global-view cluster, the administrators can easily monitor the status of other Kubernetes clusters instead of accessing each cluster individually.

In Section 3.2, we explore in detail the limitation of Prometheus Federation for monitoring multiple Kubernetes clusters and review solutions proposed for fog computing environments in the scientific literature.

Chapter 3

State of the Art

Cloud computing has received much attention from both academic and industrial communities worldwide. To meet regional requirements for applications or to avoid a single vendor lock-in, multi-cloud deployment is a powerful solution. Users can launch applications by leveraging different cloud providers in various locations. However, the cloud data centers may remain far from the end users, leading to latency issues.

Fog computing further extends the cloud computing concept with additional resources located closer to the end users. It has received much attention from academia in the last few years [123]. Many prior studies present different facets of fog/edge computing, including placement of jobs and services [124], service caching [74], [75], seamless application migration [125], [126], and supporting data stream processing [127]. These works are based on a single geo-distributed cluster, which will necessarily face scalability problems. To handle this issue, we now witness an increasing adoption of geo-distributed multi-cluster deployments. Some works focus on job scheduling [128], whereas others address resource management [129] and fault prediction [130].

In turn, the rise of multi-cluster federations for fog computing has spurred the development of various control plane solutions to manage and orchestrate federated Kubernetes environments. Multi-tenancy in federated Kubernetes clusters is also important to optimize resource utilization and ensure isolation among multiple users. We discuss the related work on multi-cluster federation and multi-tenancy frameworks designed for Kubernetes environments in Section 3.1.

Monitoring is an essential function in modern cloud data centers, which can be used to provide input for a large number of management systems such as scheduling. Fog computing has different characteristics compared to cloud computing, such as unstable network connections and geo-distribution, which create new monitoring challenges in this context. We explore the existing literature and tools in monitoring for fog computing environments in Section 3.2.

3.1 Multi-Cluster Federation Frameworks and Multi-Tenancy Frameworks

In this section, we first review the various federation frameworks proposed for Kubernetes. We then explore the solutions for multi-tenancy within Kubernetes environments. Lastly, we discuss the differences among these approaches and position our contributions in this context.

3.1.1 KubeFed and KubeFed-Related Systems

Kubernetes Cluster Federation (KubeFed) was the first system to support seamless application deployment and resource management in multiple Kubernetes cluster environments [120]. With the KubeFed control plane deployed on a Kubernetes cluster, users can centrally distribute and manage workloads across various Kubernetes clusters. In the KubeFed design, the workloads can be propagated to the different clusters by using two fields in a YAML file of a federated resource: *spec.placement.clusters* and *spec.placement.clusterSelector*. The administrators can manually select one or more clusters that should run the applications by using the *spec.placement.clusters* field. They can also utilize the *spec.placement.clusterSelector* field to let the system choose among clusters with a specific label. These manual and limited policy-based scheduling methods make KubeFed unable to scale to manage the hundreds or thousands of clusters with a large number of workloads that we expect to encounter in large-scale fog computing environments. Importantly, KubeFed does not integrate with a monitoring solution to obtain the real-time resource status in the member cluster, so it propagates workloads to target clusters without any prior checks on resource availability in the chosen clusters. This fails to manage resources efficiently, which may cause resource wastage and fragmentation. The KubeFed project is now retired and is no longer maintained or under active development [120].

Despite KubeFed limitations, the potential of using it to design geo-distributed fog computing environments was demonstrated in [131]. The authors compare two scenarios: the first one consists of a single Kubernetes cluster that manages an entire infrastructure with worker nodes located across different regions. The second scenario separates these infrastructures into independent Kubernetes clusters and federates them together using the KubeFed framework. With a cluster federation, the platform achieved a more scalable

and resilient system to face network faults, which is ideal for managing the complexities of a highly distributed scenario such as fog computing. However, the federation framework remains based on KubeFed, which therefore faces the same challenges as KubeFed.

To address the limitations of KubeFed, multi-cluster Kubernetes (mck8s) proposes to extend the KubeFed framework with resource-based automated placement, multi-cluster horizontal Pod auto-scaling, and cloud bursting [31]. It bases its placement decisions on resource utilization information, which reduces the pending Pods from 65% to 6% compared to KubeFed when executing the real-world Google cluster trace [132]. When the available computing resources are not sufficient to handle the full workload with a fixed number of member clusters, mck8s also provides a Cluster Provisioner and Cluster Autoscaler (CPCA) to automatically create new Kubernetes clusters in cloud data centers and join them in the federation. Moreover, mck8s integrates several open-source tools such as Cilium [133] and Prometheus [32] to provide additional functionality, including multi-cluster network discovery, global load balancing, and monitoring. Although mck8s addresses part of the issues from the KubeFed framework, there remain some issues with KubeFed. For example, KubeFed uses a *Push* model where the management cluster directly controls the workloads and member clusters, putting all management tasks on a single cluster. As a result, the frameworks based on KubeFed may be unable to scale to manage the hundreds or even thousands of clusters commonly required in certain multi-cluster scenarios such as fog computing. We further discuss the superiority of the Pull model compared to Push in Section 3.1.4.1.

3.1.2 Other Federation Solutions and Frameworks

To avoid a single point of failure and reach the scalability requirement of edge cloud use cases, the decentralized Kubernetes Federation Control Plane leverages distributed federated databases with Conflict-free Replicated Data Types (CRDTs) to maintain the status of resources across different Kubernetes clusters [134]. To suit this design, the system also relies on a distributed algorithm rather than a central host cluster to schedule the resources. However, this work makes the assumption that all the clusters are managed by a single entity, which may cause issues such as limited service coverage in practical fog/edge cluster deployment.

Karmada project offers solutions for managing applications across multiple Kubernetes clusters [135]. In contrast to other frameworks, it simplifies multi-cluster application management by providing a series of custom control components. Karmada is composed of

the Karmada API server, Karmada controller manager, Karmada scheduler, and Karmada agent. Detailed information for the main components is listed below:

— **Karmada API server:** This component can serve as both the interface of the control plane for Karmada and of its underlying Kubernetes, which exposes the APIs of Karmada and Kubernetes at the same time. To provide operations that are identical to the original Kubernetes, the Karmada API server is itself based on the implementation of the Kubernetes API server.

— **Karmada controller manager:** This component includes various controllers for different purposes that Karmada needs. The goal of these managed controllers is to monitor Karmada resource objects and communicate with the API servers of the underlying clusters to create related Kubernetes resources. Users can enable or disable controllers based on their requirements.

— **Karmada scheduler:** This component schedules the standard Kubernetes resources and CRD resources to the member clusters. The scheduler determines which clusters are available for the workload based on constraints and available resources. The scheduler then scores and sorts the available clusters and binds the resources to the most appropriate cluster.

— **Karmada agent:** Karmada supports both *Push* and *Pull* approaches to manage multiple clusters and applications. The Karmada agent is deployed on member clusters when using the pull model. This agent registers its representative member cluster to the Karmada control plane and pulls the manifests from the Karmada control plane to member clusters. It is also responsible for synchronizing the state of member clusters and workloads to the Karmada control plane.

However, similar to previous work, Karmada does not take into account the scenario of federate clusters managed by different organizations, which again limits the service coverage in large-scale scenarios.

Liqo is an open-source project that allows dynamic and seamless federate multiple Kubernetes clusters [136]. Liqo creates virtual node resources in the management cluster using a so-called Virtual Kubelet [137]. After a Pod is scheduled to a virtual node, the corresponding virtual kubelet creates a twin-Pod object in the member cluster for actual execution. This design ensures that all behaviors are identical to those in a single Kubernetes cluster when the administrator issues commands in the management cluster. Moreover, Liqo allows multiple management clusters to manage the same member cluster

simultaneously. However, Liqo relies on a *Push* model, which arguably limits its scalability. Moreover, all management clusters send their requests to a single shared API server in the member cluster and store the related data in a single shared database, representing only a soft version of workload isolation between different management clusters.

Open Cluster Management (OCM) presents a management model inspired by the original design principles of Kubernetes [28]. OCM separates multi-cluster operations into two phases: computation/decision (performed in the management cluster) and execution (performed in the member clusters). The management cluster stores prescriptions (i.e., the desired state of applications that users want to deploy in member clusters), whereas member clusters periodically actively *Pull* the latest prescriptions from the management cluster, ensuring that applications in the member cluster are consistent with the expected state. This design reduces the load on the management cluster and makes OCM more scalable than a push-based design. The *placement* module in OCM provides the ability to dynamically schedule the workloads to a set of member clusters. The process involves two main phases: Predicate and Prioritize, where clusters are selected based on hard requirements and then ranked based on soft requirements such as computing resources. Moreover, OCM also provides the addon framework for placement to extend the multi-cluster scheduling capabilities. For example, users can implement a customized score provider to rank the member clusters and schedule the workloads. In this thesis, we leverage OCM as our federation framework due to its *Pull* architecture nature, modularity, and extensibility. These features make it a scalable and flexible solution for our needs.

3.1.3 Multi-Tenancy Frameworks

In this section, we explore the multi-tenancy frameworks designed for Kubernetes environments. We first discuss the soft tenant isolation solutions and then review the hard tenant isolation frameworks.

3.1.3.1 Soft Tenant Isolation

By default, Kubernetes is designed for environments where all users trust each other. Supporting multi-tenancy and isolating the workloads of multiple tenants can however be realized by making each user deploy applications in a separate Namespace and by using Role-Based Access Controls (RBAC) to restrict each user in their Namespace and to scope security policies to specific Namespaces [121]. This method is considered a soft form of

isolation as all tenants share the same control plane, and appropriate configurations are required to isolate their data planes. While original Kubernetes Namespaces are useful, this method may be insufficient and inefficient for the complex needs of large organizations. Stronger forms of isolation require different techniques, as described next.

Kiosk open-source multi-tenancy framework uses flat Namespaces to provide isolated execution environments for tenant applications [138]. Kiosk defines different roles in interacting with Kubernetes. *Cluster Admin* allows one to operate and manage cluster-wide resources such as custom resources in the Kiosk framework (Account, AccountQuota, and others). Each tenant maps to an *Account*, and each *Account* is scoped to a single Namespace called a *Space* in Kiosk. This framework also provides *AccountQuota*, which is similar to the Resource Quotas [139] function in Kubernetes, to limit resource usage for *Account*. Contrary to regular Kubernetes Namespaces, each Kiosk user can only see or interact with resources in their own *Space*. Kiosk provides a *Template* mechanism to automatically deploy pre-defined resources described in a *Template*, such as network policies and Pod security policies in the designated *Space*. Although the *Template* mechanism reduces management complexity, Kiosk's flat Namespace approach may make it difficult for cluster administrators to manage numerous Namespaces. Moreover, flat Namespaces make it impossible to isolate several tenants in the same Namespace to accommodate more tenants. The project have been archived by its owner in 2024 [138].

Another project named Capsule also leverages flat Namespaces as their multi-tenancy solution [140]. Capsule improves the original Kubernetes Namespace management by grouping multiple Kubernetes Namespaces into a *tenant*. This design allows cluster administrators to manage multiple Namespaces more easily at once rather than setting policies for each Namespace separately. Although this method addresses difficult management issues, it remains a flat Namespace design that faces the same challenges mentioned above.

Extending the flat Kubernetes Namespace structure may be realized using the Hierarchical Namespace Controller (HNC) [141]. Hierarchical Namespaces allow one to apply similar policies to multiple Namespaces. By default, Kubernetes role bindings operate at the Namespace level, and each role binding must be created individually for each Namespace. HNC proposes sub-Namespaces to address this problem. The administrator can create children Namespaces of another Namespace, and the lifecycle of each sub-Namespace is bound to its parent. This design can reduce the complexity of Namespace management and further isolate specific tenants in the same Namespace. However, this remains a soft

form of tenant isolation where all tenants share a single control plane, which may cause security issues such as data leakage and cross-tenant attacks.

EdgeNet proposes another hierarchical Namespace architecture with a sub-Namespace mechanism as a way to implement a multi-tenancy mechanism [142]. There are two main differences between EdgeNet and HNC in terms of multi-tenancy: (1) EdgeNet can enforce unique names for Namespaces, whereas HNC can not; and (2) EdgeNet provides a more robust resource quota management system. For example, the HNC framework may cause uneven resource quota allocation across sub-Namespaces because it does not enforce quota setting at every level. In contrast, EdgeNet applies quotas uniformly across the entire tenant hierarchy, preventing this issue. Moreover, EdgeNet supports a federation function which then introduces the *Federation Manager* to manage the deployment of workloads from local clusters to remote clusters within a federated environment. EdgeNet also presents a custom resource called *Selective Deployment* to target a specific node or set of nodes based on specified geographic information to deploy workloads. However, EdgeNet uses a shared control plane in the remote cluster, which again faces the security issues mentioned above.

3.1.3.2 Hard Tenant Isolation

A simple way to implement hard tenant isolation is to create a separate Kubernetes cluster for each tenant or to reserve specific servers within a Kubernetes cluster for specific tenants [121]. This approach ensures that computing resources are dedicated to a single tenant, which can prevent resource contention and security vulnerabilities. However, it contradicts our goal of designing a shared fog platform for any number of tenants.

To reduce the operational complexity and cost of creating separate Kubernetes clusters, the Kamaji framework runs individual Kubernetes control plane components in Pods for achieving multi-tenancy within a single Kubernetes cluster [143]. Each tenant has its own Kubernetes control plane and dedicated worker nodes to isolate from others. This design enables centralized management of logical multiple Kubernetes clusters from a single Kubernetes cluster. In addition, this isolation approach provides tenants with a dedicated control plane that provides administrative privileges within their isolated environment. However, since each tenant has dedicated worker nodes, this solution contradicts maximizing computing resource utilization of cloud computing principals by sharing the available servers between large numbers of tenants.

The Virtual Kubernetes Clusters (vCluster) project provides a fully functional virtual cluster that runs on top of the Kubernetes cluster [29]. Each vCluster has its own control plane and schedules all workloads into the same Namespace of its control plane in the host cluster. Since each vCluster has its own API server and data store, this design provides a strong form of isolation and reduces the risk of data leakage between the tenants of different virtual clusters. Compared to the Kamaji framework, vCluster shares a pool of worker nodes among multiple virtual clusters in the host cluster by default, keeping resource usage efficient. Furthermore, the Kubernetes community has officially certified the vCluster project as a compliant Kubernetes distribution [144]. In this thesis, we use the vCluster project to handle multi-tenancy management, which is discussed in Section 4.

3.1.4 Discussion

Table 3.1 classifies the multi-cluster federation solutions and multi-tenancy frameworks by comparing parameters including sync models, support for meta-federations, multi-tenancy methods, isolation levels, and the presence or absence of certification.

3.1.4.1 Sync Models

KubeFed, mck8s, Liqo, and EdgeNet rely on the *Push* method to deploy applications to member clusters, which can be unstable in the presence of transient network failures, particularly in fog computing scenarios where network reliability can be a concern [145]. Moreover, the push model typically requires that the management cluster can access the API server of each member cluster. This may be problematic because the API servers may either be behind a firewall or not have a publicly accessible IP address.

As discussed in Section 2.4.1.3, the *Pull*-based methods that are used by Karmada and OCM are usually considered more robust, scalable, and secure. Moreover, according to the Karmada authors, the *Push* method is most suitable for deployment in public cloud environments. In contrast, the *Pull* model is better for private cloud and edge-related scenarios [146]. It provides better performance because the decentralized management by a *Pull* agent within each member cluster reduces the load pressure of the centralized control plane. In terms of each *Pull* agent is isolated in a separate member cluster, which manages its resources independently. Therefore, we consider that the *Pull* architecture is a better way to build large-scale fog computing platforms.

Table 3.1 – Comparison of multi-cluster federation and multi-tenancy frameworks.

	Sync Modes	Meta-Federations	Multi-Tenancy Methods	Isolation Levels	Certification
Multi-Cluster Federation Frameworks					
KubeFed [120]	Push	+/-	✗	✗	✗
mck8s [31]	Push	+/-	✗	✗	✗
Karmada [135]	Push/Pull	+/- (Push Mode) ✗ (Pull Mode)	✗	✗	✓ CNCF
Liqo [136]	Push	✓	Flat Namespace	Soft	✗
OCM [28]	Pull	+/-	✗	✗	✓ CNCF
Multi-Tenancy Frameworks					
Kiosk [138]	✗	✗	Flat Namespace	Soft	✗
Capsule [140]	✗	✗	Flat Namespace	Soft	✓ CNCF
HNC [141]	✗	✗	Hierarchical Namespaces	Soft	✗
EdgeNet [142]	Push	✓	Hierarchical Namespaces	Soft	✗
Kamaji [143]	✗	✗	Separate Control Plane & Data Plane	Hard	✓ Certified Kubernetes Distribution
vCluster Project [29]	✗	✗	Separate Control Plane	Hard	✓ Certified Kubernetes Distribution
Contribution					
UnBound	Pull	✓	Separate Control Plane	Hard	+/- (Certificated Components)

✓ denotes that the item is fully implemented/addressed.
+/- denotes that the item is partially implemented/addressed.
✗ denotes that the item is not implemented/addressed.

3.1.4.2 Meta-Federations with Multi-Tenancy

Liqo and EdgeNet support meta-federations because they consider the isolation between workloads being deployed by different management clusters in mind. Liqo addresses multi-tenancy by using different Namespaces in member clusters to isolate resources created by each management cluster. However, it limits itself to flat Namespaces, making it impossible to isolate further workloads produced by different users within a single management cluster. On the other hand, EdgeNet leverages hierarchical Namespace architecture, which is more flexible than flat Namespaces. However, both solutions rely on a single shared control plane and data store for all management clusters in the member cluster, which provides only soft isolation properties. Soft isolation may be risky because the management clusters may belong to owners who are different from their member clusters. This lack of hard isolation could lead to potential security and privacy issues such as data leakage between federations.

Other federation solutions make the same underlying assumption that all resources in a federation belong to a single administrative domain. They are therefore not designed to support meta-federations. These solutions basically allow users to create and propagate Kubernetes Namespaces to member clusters to isolate the workloads created by different management clusters. However, this action needs to be performed manually, which may introduce inefficiencies and possibly conflicts with the resources if multiple management clusters deploy resources to the same member cluster using the same name. Therefore, they only partially support meta-federations.

It is crucial to note that besides EdgeNet providing multi-tenancy in a cluster federation environment, other multi-tenancy frameworks do not consider the federation functionally, which limits the scope of frameworks only working in a single Kubernetes cluster.

3.1.4.3 Certification

The Cloud Native Computing Foundation (CNCF) promotes cloud-native technologies by accepting and supporting many different open-source projects to foster their development and adoption [147]. Certification from them is a critical metric for evaluating the potential and stability of the projects. Karmada and OCM have both been accepted as projects by the CNCF, which shows that they have great potential for managing multiple Kubernetes clusters [148], [149]. In the case of multi-tenancy frameworks, Capsule is also a CNCF-hosted project [150]. Kamaji and vCluster projects use the hard isolation

method that each tenant has its own sub-cluster in a Kubernetes cluster. Both solutions are certified Kubernetes distributions, which guarantees their consistency, timely updates, and confirmability [151].

3.1.4.4 Contribution

This thesis proposes UnBound, a scalable fog meta-federations platform that combines the strengths of both multi-cluster federation solutions and multi-tenancy frameworks. We leverage OCM to federate the Kubernetes clusters because of the nature of the pull model, which provides better scalability and has the potential to create very large geo-distributed fog federations. UnBound then specifically addresses the multi-tenancy issue for supporting meta-federations, where individual fog Kubernetes clusters may lease their resources to multiple administrative domains by using the vCluster project. vCluster project uses separate control planes with a shared data plane, which can provide hard isolation for inter-federation while keeping high resource utilization within the member clusters. The code base of UnBound is small thanks to the usage of existing open-source projects. We consider this as a strength of UnBound which can facilitate broad adoption by the Kubernetes community as well as long-term maintenance and support. Moreover, we plan to make UnBound available in open-source, which ensures compatibility and reliability for users while fostering community collaboration and continuous improvement. Although UnBound is based on Kubernetes and its ecosystem, the concepts of the UnBound platform hold broader applicability and can in principle be applied to current or future container orchestrators, multi-cluster federation solutions, and multi-tenancy frameworks.

3.2 Monitoring for Fog Computing Environments

Monitoring is an essential functionality in current computing environments. There are several purposes for monitoring, including resource usage, fault detection and diagnosis, billing, and performance monitoring. Among these, the main objective of geo-distributed monitoring is to track the resource usage of computing nodes, particularly in potential resource-restricted fog environments. A number of open-source and commercial monitoring tools such as DARGOS [152], Zabbix [153], PCMONS [154], JCatascopia [155], and Nagios [156] were developed to suit cloud computing requirements. However, they are not considered appropriate for geo-distributed fog computing environments [157]–[161]. On the other hand, to overcome the challenges of monitoring in a fog computing envi-

ronment, some authors present monitoring solutions and architectures designed with the specific constraints of fog computing in mind.

3.2.1 Monitoring Solutions for Fog Computing

PyMon provides a monitoring solution for fog environments specially designed to run on ARM-based single-board computers [162]. It collects monitoring data from devices at a periodic rate and sends them to a centralized PostgreSQL database. PyMon also offers a web interface based on the Django framework and can show graphs and tabulars of the monitored system status. To collect the monitoring data, PyMon reuses the Monit lightweight open-source software [163]. Although PyMon is a lightweight monitoring solution that is suited for resource-restricted environments, its scalability was not evaluated. This means it may not handle large-scale scenarios [159]. Moreover, PyMon is not adaptive and flexible, which does not support on-the-fly configuration changes or data transmission frequency modification [157].

FMonE aims to address monitoring challenges in fog environments with an independent, stand-alone solution in fog environments [164]. It relies on a container orchestration system called Marathon [165] to build the monitoring workflow based on the user requirements. The system gathers monitoring data at a periodic interval through a centralized or hierarchical structure. FMonE combines pull and push methods for data collection and can monitor infrastructure, platforms, and services. However, the authors evaluate their work using up to 78 Virtual Machines (VMs). This number remains very far from the scale at which global fog platforms are expected to operate.

FogMon proposes a lightweight and Peer-to-Peer (P2P) monitoring architecture that deploys an agent in each member node called "Follower." Followers report hardware-based metrics and network QoS data to their "Leader" node [166]. Each Follower node is linked to a single Leader node and runs in a classic client-server model. Follower nodes periodically push data to a Leader node. The Leader node aggregates the monitoring data and disseminates them to other Leader nodes using a gossip protocol. To reduce the network traffic between Followers and Leaders, FogMon adopts a solution where Followers only send data with the average or variance value greater than a threshold compared to the last sent. In addition, FogMon has been refined into FogMon2, which adds new features and improves handling of latency and bandwidth degradation [167]. FogMon2 was evaluated in the Fed4Fire testbed with up to 40 nodes. However, this number of nodes is still too limited to prove its scalability.

AdaptiveMon extends FogMon with a self-adaptive monitoring solution for fog environments. It introduces two additional functions: Indicators Selection and Change Rate [168]. Indicators Selection reduces the number of metrics, whereas Change Rate adjusts the frequency of metrics reported from Follower to Leader. Using these two features and comparing them to the original FogMon framework, the results show that AdaptiveMon can save energy and reduce network I/O, with the trade-off of requiring more memory resources. The authors conduct the experiments in a Linux virtual machine with Docker containers as nodes (one Leader and one Follower), which only focus on individual peers. However, AdaptiveMon, an extension of FogMon, still needs to prove its scalability and suitability for integration with modern orchestration frameworks such as Kubernetes federation [157].

DEMon is a decentralized and self-adaptive monitoring framework specially designed for edge environments [169], [170]. It does not rely on a single point of control for storing data and controlling the system. DEMon uses a gossip-based protocol to disseminate the monitoring data to other edge nodes in the system. Moreover, each edge node can be self-adaptive by adjusting monitoring settings to balance monitoring data quality with resource usage. DEMon also proposes a Leaderless Quorum Consensus (LQC) protocol to retrieve the monitoring data for users or client applications. DEMon was examined in a large-scale simulated edge environment with up to 300 nodes and a real-world testbed with 12 Raspberry Pi nodes. The results demonstrate that DEMon effectively shares and retrieves monitoring data and proves its scalability, which suits edge environments.

The most popular monitoring tool is Prometheus [32]. It has been accepted by the Cloud Native Computing Foundation (CNCF) as a "graduated" project, which shows its great potential in conjunction with the de-facto standard Kubernetes container orchestrator and demonstrates that Prometheus is a stable and production-ready monitoring system [25]. At the same time, many research works use Prometheus as a basis for system monitoring [171]–[176]. Prometheus provides a function called "Federation" which allows a Prometheus server to collect metrics from other Prometheus servers. A common use case is building a global-view Prometheus server, which scrapes and stores the monitoring data of other Prometheus servers. Two levels of federation are instance-level drill-down and job-level drill-down. In Prometheus terminology, an *instance* is an endpoint that the user can scrape from, and a *job* is a collection of *instances* with the same purpose. Prometheus Federation has been used to monitor systems in numerous studies [18], [31], [177], [178].

3.2.2 Issues of Prometheus Federation

However, except for the Prometheus monitoring system, other monitoring solutions are discussed at the node level, which are not designed for cluster federation environments where nodes are not considered individually but cluster by cluster. Considering the graduated maturity level and functionality of the monitoring frameworks, this thesis leverages Prometheus and its Prometheus Federation function as the monitoring solutions for cluster federation. However, the Prometheus Federation also has a number of limitations that this thesis aims to address.

(1) The highest scrape level of Prometheus Federation is job-level, and it uses the *match* mechanism to select the series of metrics. For example, the operator can write job="Node-exporter" in a federation server's configuration file to scrape the metrics that match this label from the target Prometheus servers. It results in scraping the matching metrics that are all the nodes[1] in the target cluster when job="Node-exporter" is set. This design is suitable for backing up metrics for high availability purposes but not for letting a management cluster manage federated clusters. It wastes network bandwidth to transmit and disk resources to save the same node metrics in the management cluster.

(2) Prometheus Federation appends all original labels in each metric when a Prometheus server scrapes from the target Prometheus server to identify where the metric comes from. However, not all original labels are necessary for recognition, and the scheduler may not need this detailed information to make scheduling decisions. Furthermore, the labels are attached before the metrics transmission, which increases the cross-cluster network traffic.

(3) Prometheus scrapes all the metrics even if some of the metrics values did not change. This unnecessary data transfer would waste network bandwidth, particularly when dealing with a large number of targets across a large fog federation.

(4) Prometheus Federation collects the monitoring data from target clusters at a fixed periodicity. If the member clusters are mostly idle, the federation 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 reducing the cross-cluster network traffic. However, when computing resources in member clusters are in high

1. We assume all nodes in all clusters have installed Node-exporter and labeled job="Node-exporter".

demand, minimizing the number of pending Pods [2] in member clusters requires one to carefully determine which clusters have enough resources to run the applications. 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.

To overcome these monitoring challenges and optimize the value of scrape interval, we base our work on the Prometheus monitoring ecosystem and introduce Acala and AdapPF. Both aim to balance cross-cluster network traffic and the accuracy of monitoring data. Acala automatically aggregates the metrics whose metric name and labels are identical in different servers, which reduces the cross-cluster network traffic as well as the deployment and configuration cost. This addresses the issues (1) and (2). Acala also deduplicates metrics values and thereby avoids transferring unchanged values over and over again to address the issue (3). To understand the impact of scrape interval, cross-cluster network traffic, and data accuracy while avoiding the accuracy effects introduced by metrics aggregation, we propose AdapPF which can dynamically adjust the values of the scrape interval in a geo-distributed cluster federation environment. It checks the current resource status of the target cluster with non-aggregated data and then adjusts the scrape interval to address the issue (4). We argue that these two solutions complement each other and could be combined in principle.

2. In the event that there are insufficient resources on the nodes within the member cluster to start the Pod, the Pod gets placed in a pending state until adequate resources become available.

Chapter 4

Multi-Tenancy Management in Scalable Fog Meta-Federations

4.1 Introduction

Enabling fog platforms to embrace the full benefits of cloud computing principles requires the design of large-scale, multi-tenant, geo-distributed fog computing platforms that any application may make use of and where statistical multiplexing of large numbers of independent workloads can help guarantee high resource utilization. This chapter proposes the design of scalable fog *meta-federations* to address this challenge. We define meta-federations as a complex ecosystem composed of many independent fog resource providers that may set up business agreements with one another to allow access to their computing resources and thereby expand their geographical span in locations where they do not own resources themselves. We discuss the concept of meta-federations in detail in Section 4.2.

An important and difficult challenge of fog meta-federations is multi-tenancy. In these systems, the same group of servers may be used to host workloads belonging to multiple tenants who are customers of different providers. This scenario requires the system to guarantee isolation at two different levels. First, two tenants of the same fog provider should not be able to see or interfere with each other's workloads. Second, two fog providers that have established access to the same member cluster should also not be able to see or interfere with one another. To our best knowledge, this two-level multi-tenancy challenge is unique to federated environments where the same cluster may belong to multiple independent federations.

The second challenge in designing fog meta-federations is that of scalability. Building a fog computing infrastructure at the scale of a country or even a continent requires one to aggregate resources in thousands of different locations. A single "management cluster" therefore needs to be able to control access to thousands of "member clusters." Conversely,

to maintain high resource utilization, each member cluster may decide to join numerous independent federations, each with its own management cluster.

In this chapter, we propose UnBound, a scalable fog meta-federations platform that specifically addresses the multi-tenancy scenario, where individual fog clusters may lease their resources to multiple administrative domains. UnBound relies on Kubernetes to orchestrate resources within individual fog clusters [27] and Open Cluster Management (OCM) to federate multiple member clusters under the authority of a management cluster [28]. We address the issue of multi-tenancy management by isolating federations within a single member cluster using the Virtual Kubernetes Clusters (vCluster) [29] project to create isolated logical sub-clusters within the member clusters. Each vCluster [1] has its own API server and data store, which provides stronger isolation guarantees than simple Kubernetes Namespaces to ensure that different federations do not interfere with one another.

We conduct extensive evaluations through real-world deployments in the Grid'5000 testbed [30] and demonstrate that UnBound achieves inter-user and inter-federation isolation while maintaining comparable application creation time to the original Open Cluster Management and avoiding increasing cross-cluster network traffic between the management and member clusters. Moreover, the resource consumption of UnBound components remains within acceptable limits. Finally, we demonstrate the stability and scalability of UnBound using federations with up to 500 member clusters and a member cluster belonging to up to 100 independent federations.

The remainder of this chapter is organized as follows: Section 4.2 discusses the motivation behind this work. In Section 4.3, we describe the design and components of UnBound meta-federations. We evaluate our solution in Section 4.4 and summarize the chapter's conclusions in Section 4.5.

Parts of this chapter were published in [179].

4.2 Motivation

Multi-cluster federations are a standard technique to aggregate the resources of multiple independent Kubernetes clusters into a single logical entity [117]. This concept enables users to gain seamless access to a large computing infrastructure in multiple geographical

1. We use the term "vCluster project" to refer to the entire vCluster framework, and the term "vCluster" to refer to a virtual cluster created for isolation between different management clusters in a member cluster.

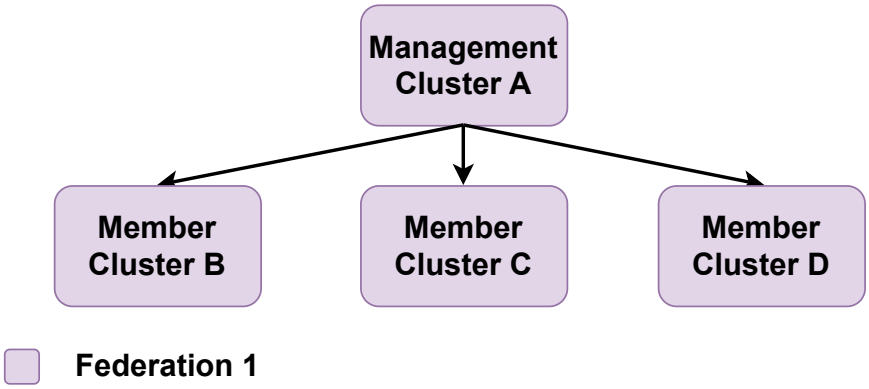


Figure 4.1 – An example of KubeFed architecture. Management Cluster A manages multiple Member Clusters B, C, and D.

locations. Federation users submit their workload deployment requests to a single *management cluster*, which subsequently forwards requests to one or several of their *member clusters* according to the job's metadata and some pre-defined scheduling policies. However, as shown in Figure 4.1, standard federation frameworks such as KubeFed [120] basically follow a hierarchical organization where a given cluster should be either a management cluster or a member of a single federation. More complex organizations (e.g., a cluster being a member of two different federations while also being a manager of a third one) are usually not supported. We refer the reader to Section 3.1.4.2 for a detailed discussion. Also, traditional federations assume that all the hardware resources in a federation belong to a single administrative domain, which limits federations to scenarios where the organizations that own the resources have total trust in each other.

We propose a different model where multiple clusters belonging to different organizations may freely establish or revoke peering relationships with one another. Figure 4.2 shows an example where three federations co-exist with varying types of relationships between the clusters. Federation 1 (in purple color) is a geo-distributed federation where management cluster A has established access rights to member clusters C, D, and E. Cluster E has multiple roles because it is also a member of Federation 3 while being in charge of managing Federation 2. Federation 2 also expands to multiple other member clusters. This example demonstrates the versatility of meta-federations capabilities, accommodating various configurations, including the one-to-many and many-to-one relationships showcased here.

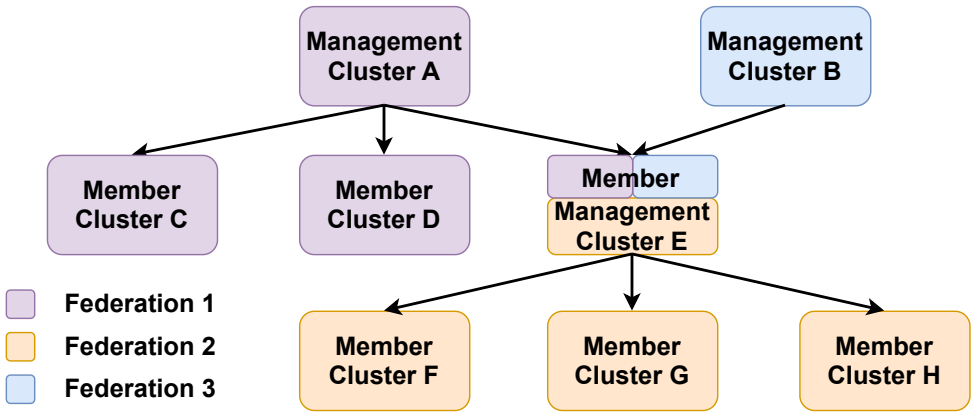


Figure 4.2 – An example of meta-federations.

In meta-federations, each cluster may be owned by a different entity. Peering relationships between clusters may be established via a legal contract where one company leases some of its hardware resources to another, following a classical cloud-like business model. For example, Cluster C and Cluster D may belong to two separate companies in different locations. Both of them have established a peering contract with Federation 1 so that Federation 1 can use their computing resources and expand its range of service locations.

4.3 System Design

In this section, we propose UnBound, a scalable fog meta-federations platform that considers different levels of multi-tenancy to support clusters from different organizations in multiple Kubernetes cluster environments.

4.3.1 System Model and Meta-Federations

Considering the characteristics of fog computing, we assume that a fog federation may be composed of multiple Kubernetes clusters. These Kubernetes clusters may belong to different organizations or companies and may be located in various regions. Within each cluster, we assume that all servers are located in the same geographical location and that all clusters and nodes can communicate with each other through the network infrastructure. We further assume that each Kubernetes cluster has enough computing resources to run UnBound's components.

In meta-federations, fog resource providers can establish business agreements to share their computing resources with one or more federations. These agreements allow Kubernetes federations to expand their geographical reach and serve users in different areas by using computing resources from other Kubernetes clusters. Different organizations may operate their own federations and host multiple users in their federations.

Users can choose to run their applications on a specific Kubernetes cluster or to distribute them among member clusters of a given federation. As a result, two separate isolation scenarios arise: inter-user isolation and inter-federation isolation. Inter-user isolation requires isolation between different users within the same Kubernetes cluster in a non-federated environment. Once a Kubernetes cluster takes the role of management cluster of a federation, the users in this management cluster can deploy applications across all member clusters in the federation. Similar to the case of a single cluster, the isolation of the users in the member clusters must also be considered. For inter-federation isolation, each federation, which may belong to different fog providers, should also not be able to see or interfere with one another and should be isolated when these federations access the same Kubernetes clusters.

We expect fog computing platforms to be geographically distributed, with their member clusters spanning a country or even a continent. In this environment, the scalability of meta-federations is a critical challenge. Each management cluster should be able to manage a large number of member clusters simultaneously, and each member cluster should be able to join multiple management clusters from different organizations to maintain high resource utilization. As a result, meta-federations support both *one* management cluster to *many* member clusters and many-to-one configurations. Therefore, each Kubernetes cluster can concurrently operate as both a management cluster and a member cluster.

4.3.2 System Architecture

To achieve the vision of meta-federations, UnBound relies on Kubernetes to orchestrate resources within individual fog clusters. Then, we build UnBound meta-federations based on two open-source projects, Open Cluster Management (OCM) and Virtual Kubernetes Clusters (vCluster). OCM is responsible for managing federated clusters and distributing the workloads across these clusters, where the *Pull* model of OCM is well-suited for large-scale federations to address the scalability challenge of meta-federations. To ensure the isolation between different federations in the same member cluster, we select the vCluster

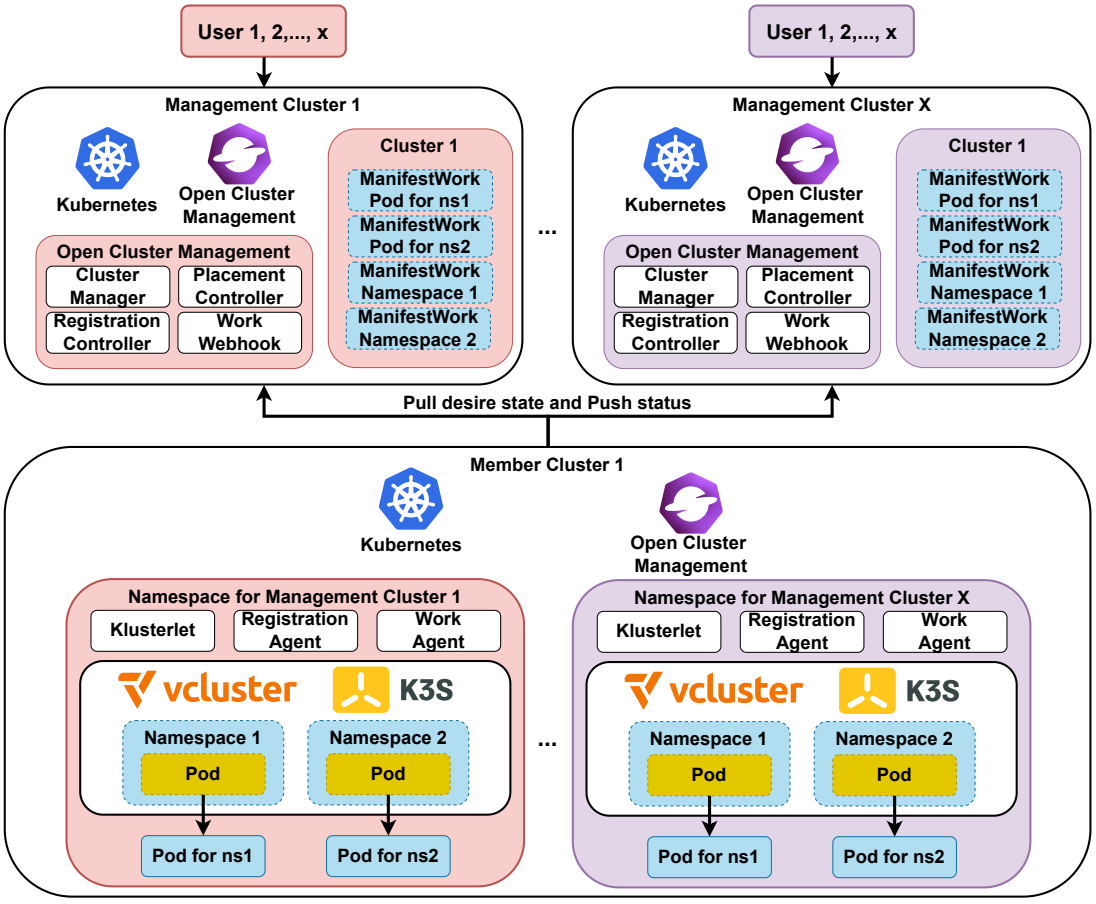


Figure 4.3 – Architecture of UnBound where a member cluster shares its resources with two independent management clusters. The "ns" abbreviation stands for Namespace.

project as our multi-tenancy solution because of its hard isolation guarantees and low performance overhead.

Figure 4.3 illustrates the UnBound architecture and its components. A management cluster can accommodate multiple users who can utilize this Kubernetes cluster to deploy workloads in the host cluster or member clusters via the `kubectl` or `clusteradm` [2] tools. After a member cluster joins the management cluster, the system provisions a dedicated Kubernetes Namespace to represent the member cluster. Users can manually create ManifestWorks to this Namespace in the management cluster (we discuss ManifestWork in the following section), and the system then deploys the workload, which is described in ManifestWorks, to the corresponding member cluster. Moreover, users can also distribute

2. The `clusteradm` command-line interface allows users to interact with Open Cluster Management clusters.

workloads with the placement-controller. Based on the requirements of ManifestWorks, the placement-controller selects a set of member clusters and deploys the ManifestWorks to the dedicated Namespaces which represent the target member clusters in the management cluster.

Within each member cluster, UnBound creates a specific Kubernetes Namespace for each corresponding management cluster and installs the OCM components and k3s-based vCluster components in this Namespace. The work-agent in the member cluster continuously monitors the Namespace from the management cluster, pulling the latest ManifestWorks states, synchronizing them with the respective vCluster through its own API server, and pushing the current status of workloads to the management cluster. The metadata of workloads from the management cluster are stored in the vCluster's own data store. Based on these metadata, vCluster creates corresponding Pods or related Kubernetes resources in the underlying host Kubernetes cluster in the same Namespace of OCM components and vCluster. This design isolates all UnBound components and workloads from a management cluster within a dedicated Namespace in a member cluster, preventing interference from other management clusters. Additionally, this approach facilitates the enforcement of resource quotas in the Namespace for each management cluster within the member cluster [139]. We leave the topic of dynamically setting the resource quotas of different Namespaces and possibly allowing quota oversubscription for future work.

Note that the Kubernetes Namespace isolates the Pods created by different management clusters. This design results in two types of isolation. Pods created within a single federation are isolated using Namespaces in the member cluster, using the same mechanism as in the original Kubernetes. Furthermore, vCluster isolates the metadata and the workload requests of each federation with its own storage backend and API server.

4.3.3 Components of UnBound

This section discusses details of the main UnBound components.

4.3.3.1 Open Cluster Management

Open Cluster Management (OCM) [28] simplifies the management of multiple Kubernetes clusters by decomposing multi-cluster operations in two phases: computation/decision and execution. Consequently, a federation is composed of two different roles: management cluster (hub) and member cluster (agent).

A federation's management cluster is responsible for managing and controlling multiple member clusters. The management cluster also makes placement decisions to distribute the workloads across the member clusters. On the other hand, each member cluster is responsible for carrying out the management cluster's instructions and running the workloads that were assigned to it.

To achieve greater scalability for the federation platform, OCM employs the "hub-agent" architecture, which mirrors the original "hub-kubelet" pattern from Kubernetes. This architecture utilizes a *Pull* mechanism to retrieve the latest prescriptions from the management cluster and to continuously reconcile the workloads to the desired state in the member cluster.

OCM introduces a Custom Resource (CR) called ManifestWork [110], [111]. A ManifestWork defines a group of Kubernetes resources to be deployed across member clusters in a federation. Users deploy ManifestWorks in a particular Namespace in the management cluster, and the work-agent in the member cluster subsequently monitors the contents of ManifestWorks in that Namespace to keep the status of workloads in sync with it. Note that only workloads that use ManifestWork will be deployed to the member clusters.

In the management cluster, UnBound exploits four main components from OCM: cluster-manager, placement-controller, registration-controller, and work-webhook. We discuss each component below.

— **Cluster-manager:** The cluster-manager serves as an operator responsible for establishing and managing other components within the management cluster.

— **Placement-controller:** The placement-controller enables users to schedule their workloads to a set of member clusters automatically. The OCM scheduling framework is based on the Kubernetes scheduling architecture and is organized in two steps: predicate and prioritize. The predicate handles hard requirements, such as label selection and taints/tolerations. After selecting clusters that satisfy the mandatory hard requirements, the prioritize phase evaluates the clusters identified in the predicate step based on soft requirements such as the number of clusters and resource status of clusters, to determine a suitable subset of member clusters. Moreover, users can expand the multi-cluster scheduling functionality through the OCM add-on framework.

— **Registration-controller:** Two main tasks for the registration-controller are for member cluster registration and receiving the health status of member clusters.

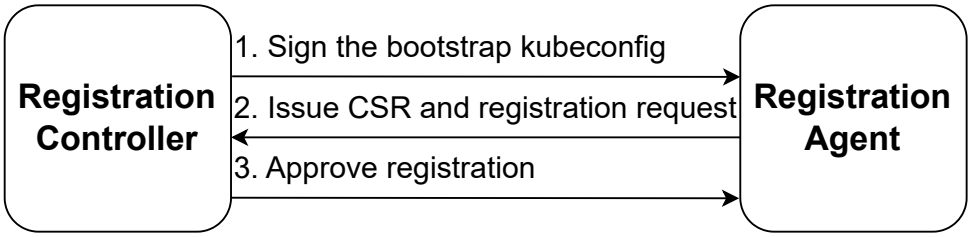


Figure 4.4 – Registration process between one management cluster and one member cluster, with registration-controller in the management cluster and registration-agent in the member cluster.

— **Work-webhook:** The work-webhook is an admission webhook running in the management cluster to validate the content of ManifestWorks.

UnBound leverages three OCM components in the member clusters: Klusterlet, registration-agent, and work-agent.

— **Klusterlet:** The task of Klusterlet is similar to cluster-manager, which is a bootstrap application to create and manage other agents in the member cluster.

— **Registration-agent:** The registration-agent operates in the member cluster and handles the registration process. Figure 4.4 shows the three steps of registration between a management cluster and a member cluster. First, the registration-controller gets the bootstrap token so that the user can use this token to join the management cluster in the member cluster. Then, the registration-agent issues a Certificate Signing Request (CSR) and sends it to the controller. Finally, the administrator in the management cluster approves the request for registration. After the registration procedure, the registration-agent keeps regularly sending heartbeats to the controller and checking the certificate of registration validation.

— **Work-agent:** The work-agent monitors the ManifestWorks in a Namespace that represents the work-agent hosted member cluster in the management cluster. When the work-agent detects a change in the Namespace in the management cluster, it applies or changes the Kubernetes resources included in the ManifestWorks to the member cluster.

4.3.3.2 Virtual Kubernetes Clusters

Virtual Kubernetes Clusters (vCluster) project aims to run a full Kubernetes cluster as an application and host it in another Kubernetes cluster to provide multi-tenancy

functionality [29]. It supports different Kubernetes distributions, including vanilla Kubernetes, K3s, and K0s. Each vCluster has its own control plane, providing better tenant isolation than the original Kubernetes Namespaces. Moreover, vCluster has its own storage backend and supports different databases, such as SQLite, MySQL, PostgreSQL, and etcd. vCluster relies on its host underlying cluster to provide its own worker node pool and networking resources.

The vCluster project separates Kubernetes resources using two levels: high-level and low-level. High-level resources, such as Deployment, StatefulSet, and Custom Resource Definitions (CRDs), are purely virtual. These resources only reach the vCluster API server and get stored in vCluster's data store to avoid using the API server and data store from the underlying Kubernetes cluster. Since vCluster does not have actual worker nodes and networking, some "low-level" resources such as Pods and Services need to be synchronized to the underlying Kubernetes cluster and in the same Namespace as the vCluster.

UnBound leverages vCluster to isolate different management clusters, which may make simultaneous usage of the same member cluster. UnBound therefore deploys vCluster only in the member cluster. Although vCluster can also run in the management cluster for different users, we leave this topic for future work.

Each vCluster has two components: control plane and syncer.

— **Control plane:** vCluster's control plane includes the Kubernetes API server, data store, controller manager, and optional scheduler. In UnBound meta-federations, the Kubernetes API server handles the requests from the work-agent to create, update, or delete the workloads in the vCluster. The data store is the database where the API server stores metadata of all resources. The controller manager monitors the status of the entire vCluster and ensures that the vCluster is in the expected working state. The administrator can enable the scheduler inside the vCluster to provide scheduling with custom requirements, such as affinity and topology spreading in the host Kubernetes cluster.

— **Syncer:** vCluster uses a syncer to create low-level resources, such as Pods and Configmaps, in the underlying host cluster. To schedule low-level workloads in the host cluster, vCluster reuses the host cluster's scheduler to place workloads by default. As Pods are scheduled directly in the underlying host cluster, they experience no performance degradation. Similar to the work-agent component of OCM, after deploying low-level resources, vCluster's syncer keeps periodically synchronizing the status between vCluster's control plane and the underlying host cluster.

4.3.3.3 Multi-Tenancy Management

UnBound meta-federations support two levels of isolation: inter-user and inter-federation. We discuss these two levels in detail below. We divide inter-user into two cases: users within a cluster and users within a federation.

— **Inter-user multi-tenancy within a Kubernetes cluster:** Kubernetes cluster administrators can use Kubernetes *Namespaces* to isolate workloads from each other. To limit resource usage in different Namespaces, the administrators can also leverage the resource quotas function [139] to divide cluster computing resources between multiple users in each Namespace.

— **Inter-user multi-tenancy within an UnBound meta-federation:** When a Kubernetes cluster becomes the management cluster of an UnBound meta-federation, its administrator can create Kubernetes *Namespaces* in the vCluster using Manifestworks (see Figure 4.3). The capability to isolate different users within the federation stems from the vCluster functionality which allows users to create cluster-scoped resources such as Namespaces. Furthermore, unlike the original OCM, which directly accesses Kubernetes clusters where the control plane manages the entire cluster, the completely separate control plane of vCluster offers users access to vCluster for detailed management or debugging tasks through the vCluster's kubeconfig file without affecting other vClusters or the host Kubernetes cluster.

— **Inter-federation multi-tenancy within a single member cluster:** Management clusters potentially belonging to different organizations can concurrently use the computing resources of the same member cluster. To enable multi-tenancy management in this scenario, UnBound employs vCluster to isolate workloads associated with different management clusters. Each vCluster uses its own control plane, which effectively isolates workloads from different management clusters in the same member cluster. Moreover, the metadata of the Kubernetes resources in vCluster are stored in its own data store. As a result, requests and metadata created by management clusters cannot reach the underlying Kubernetes cluster nor other vClusters, which provides a stronger form of isolation than Kubernetes Namespaces. Note that the owner of a member cluster can still access information about all workloads in their cluster. Therefore, while the owner lends its clusters to others, it can still enforce the terms of the resource leasing contract.

4.4 Performance Evaluation

We evaluate UnBound using four sets of experiments: (i) multi-cluster application creation in a member cluster; (ii) application stability despite a vCluster failure; (iii) one management cluster with multiple member clusters; and (iv) multiple management clusters with one member cluster. We run the experiments in both cloud and fog networking environments.

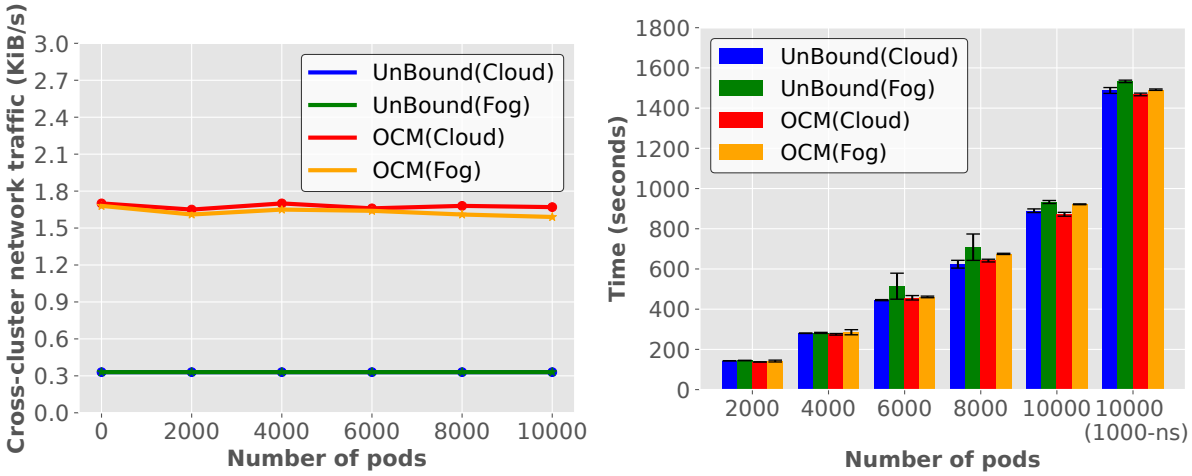
4.4.1 Experimental Setup

We base our prototype implementation on the principles outlined in Section 4.3. We modify the code from the Open Cluster Management project to automatically create vCluster in the particular Namespace and connect the work-agent to vCluster. We also fine-tuned the vCluster configurations to suit UnBound meta-federations. We use Kubernetes v1.27.3, Open Cluster Management v0.11.0, vCluster v0.15.5 with k3s v1.27.3, and Cilium v1.13.4 for the deployment. For data collection, we use the tcpdump package to collect the cross-cluster network traffic and Kubernetes Metrics Server v0.6.4 to measure resource consumption, including CPU and memory usage [180]. We then compare our solution to the original Open Cluster Management.

To ensure that our experiments closely resemble production conditions, we run our evaluations in the Grid'5000 geo-distributed testbed [30]. To emulate a fog networking environment, we use the netem [181] package to introduce a 50 ms delay with 5 ms jitter to both the network ingress and egress, resulting in a total one-way network latency of 100 ms with 10 ms jitter.

4.4.2 Multi-Cluster Application Creation in a Member Cluster

In this experiment, we launch two Kubernetes clusters: a management cluster that uses a single Virtual Machine (VM) with 16 cores and 32 GiB of memory; and a member cluster that uses 101 VMs, including one control plane node with 16 cores and 32 GiB of memory and 100 worker nodes with 2 cores and 4 GiB of memory each. We deploy up to 1000 ManifestWorks in the management cluster to create Kubernetes Deployments in the member cluster. Each Deployment creates 10 nginx Pods, resulting in up to 10,000 Pods in total within the same Namespaces. Moreover, we also create a scenario with 10,000 Pods where each Deployment runs in its own Namespace in the member cluster so there



(a) Cross-cluster network traffic (collect in the management cluster).

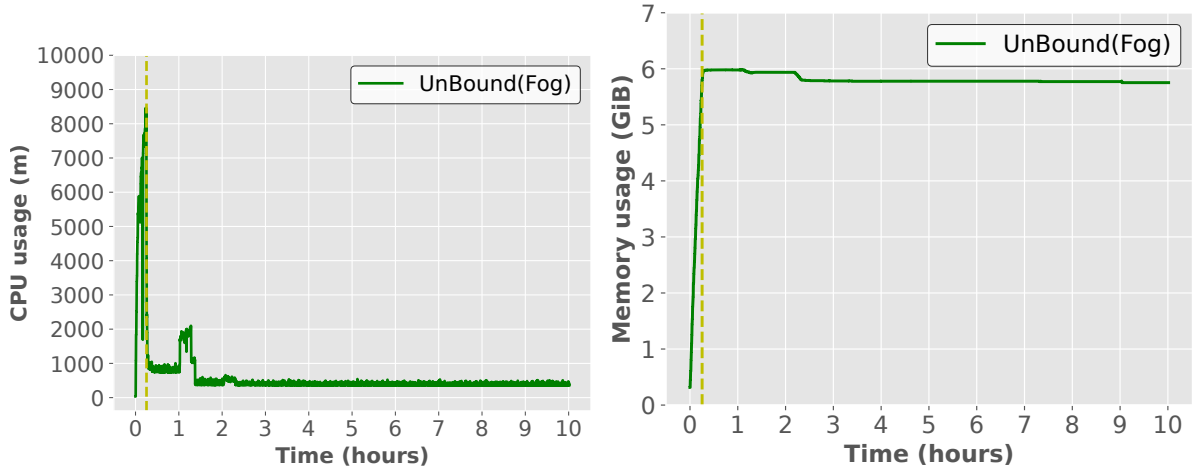
(b) Application creation time.

Figure 4.5 – Cross-cluster network traffic and application creation time in multi-cluster application creation experiment.

are a total of 1000 Namespaces, and each Namespace contains 10 nginx Pods to simulate inter-user isolation in UnBound. We refer to this scenario as 1000-ns in the following section.

We run each experiment three times and report the average results across them. The only exceptions are Figure 4.6 for long-term collection experiments, which show the results of a single run (repeating these experiments draws similar results). For other figures, we skip the first 9000 seconds of data and average the remaining 7200 seconds as the results for a round. We do this because we found that vCluster optimizes deployed resources and uses slightly more computing resources during the first 9000 seconds, as we can see in Figure 4.6. Therefore, the data during this first period does not represent normal resource usage.

Figure 4.5(a) demonstrates that UnBound effectively reduces cross-cluster network traffic in both cloud and fog environments for any number of Pods. The average network traffic across different numbers of Pods in UnBound is 0.33 KiB/s in both environments, while the average network traffic for OCM is 1.68 KiB/s in the cloud and 1.63 KiB/s in the fog. To enhance clarity, the figure excludes the results for 1000-ns experiments, which are essentially identical to those for 10,000 Pods within a single Namespace. These findings not only show that our solution does not introduce additional cross-cluster network



(a) CPU usage of vCluster in 10,000 Pods case (single Namespace) with long-term collection.

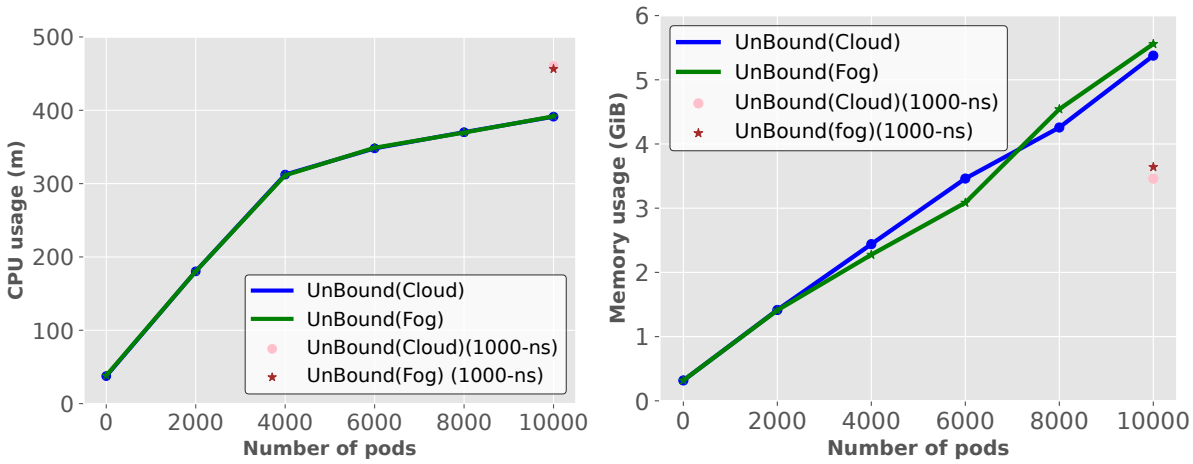
(b) Memory usage of vCluster in 10,000 Pods case (single Namespace) with long-term collection.

Figure 4.6 – CPU and memory usage of vCluster with long-term collection in multi-cluster application creation experiment. The yellow dash lines indicate the time when all Pods reach running status.

traffic between the management and member clusters but also evidence the effectiveness of UnBound.

Figure 4.5(b) presents the time it takes to create applications from the management cluster to the member cluster. We measure the creation time by the time of script execution and then wait until all Pods reach running status in the member cluster. We can see that the creation times for UnBound(Cloud), UnBound(Fog), OCM(Cloud), and OCM(Fog) are 1488, 1533, 1468 and 1492 seconds, respectively, for the 10,000 Pods case. The fog environment takes a little longer to finish, regardless of whether the method is UnBound or OCM. This result is due to network latency, which may cause a delay in pulling the latest prescription. Our solution takes 20 (Cloud) and 41 (Fog) seconds longer than OCM in the same condition, which shows that the overhead of our approach is small. The 1000-ns scenario shows the same trend, but it takes longer to create the applications because the Namespaces for each Deployment must also be created, resulting in 2000 ManifestWorks instead of 1000 in the same Namespace case.

Figure 4.6(a) and Figure 4.6(b) represent the long-term collection of 10,000 Pods in the same Namespace in the member cluster. We only show the results for the fog environment, as the results for the cloud environment are similar. Before 914 seconds, the member cluster is busy creating Pods, which causes CPU and memory usage to increase. Once all



(a) CPU usage of vCluster.

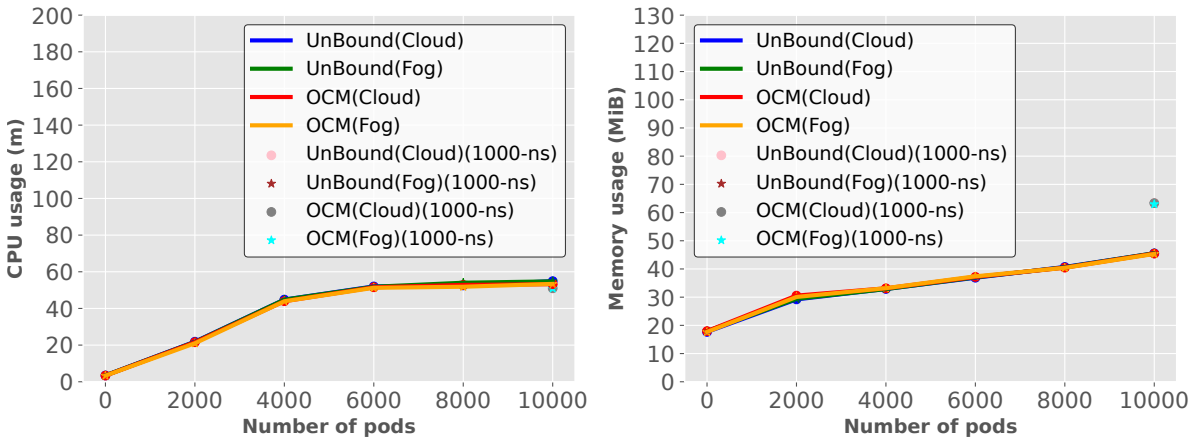
(b) Memory usage of vCluster.

Figure 4.7 – CPU and memory usage of vCluster in multi-cluster application creation experiment. 1000-ns represents the results of a scenario with 10,000 Pods that each Deployment runs in its own Namespace, resulting in 1000 Namespaces.

Pods are running, the vCluster spends approximately 2.5 hours optimizing the deployed workloads, which requires more CPU resources than the maintenance phase. Memory usage follows the same trend, stabilizing after the vCluster completes its optimization. The average CPU usage in the maintenance phase is 391 milli-cores with a standard deviation of 36 m [3], and the average memory usage is 5.9 GiB with a standard deviation of 9 MiB. These results show that the resource usage of vCluster is small and stable.

Figure 4.7(a) demonstrates vCluster CPU usage with different numbers of Pods. The results show that the CPU usage trend is almost the same in both cloud and fog environments, up to 391 m in the 10,000 Pod case. For the 1000-ns scenario, vCluster consumes more CPU usage, 460 m (Cloud) and 456 m (Fog), because it maintains more resources, including Namespaces, Kubernetes Deployment, and Pods. As shown in Figure 4.7(b), the memory usage of vCluster in the cloud environment experiences linear growth from 0.3 GiB (for 0 Pod), 1.4 GiB (2000 Pods), to 5.4 GiB (10,000 Pods). However, the memory usage for the 1000-ns scenario consumes less memory usage, 3.5 GiB (Cloud) and 3.6 GiB (Fog). According to the results, UnBound can efficiently utilize resources in both cloud and fog environments. The CPU and memory usage are relatively low, even when the number of Pods reaches large values.

3. The Kubernetes metrics server uses millicores (m) to measure CPU usage. One millicore equals 0.001 vCPU/core for cloud providers or 0.001 hyperthread on bare-metal Intel processors [182].



(a) CPU usage of work-agent.

(b) Memory usage of work-agent.

Figure 4.8 – CPU and memory usage of work-agent in multi-cluster application creation experiment. 1000-ns represents the results of a scenario with 10,000 Pods that each Deployment runs in its own Namespace, resulting in 1000 Namespaces.

Figure 4.8(a) illustrates the work-agent resource usage. The CPU usage is similar regardless of the methods or environments. For 10,000 Pods, the CPU usage is between 51 m and 55 m. Figure 4.8(b) shows that the 1000-ns case uses more memory, which is around 63 MiB, whereas the memory usage of 10,000 Pods in a single Namespace is 45 MiB.

Overall, these results demonstrate that UnBound performs comparably to the original Open Cluster Management in both cloud and fog network environments despite introducing additional components to provide inter-federation isolation.

4.4.3 Application Stability Despite a vCluster Failure

UnBound leverages vCluster to isolate different federations in the member cluster. However, vCluster is an additional component that may become a weak point, as vCluster may crash and affect the workloads they manage. To demonstrate the resilience of UnBound in the face of vCluster failures, we launch two clusters, one serving as the management cluster and the other as a member cluster. In this experiment, we use Online Boutique, a micro-service demo application [183]. We create ManifestWorks of this micro-service demo in the management cluster and wait for these applications to reach running status in the member cluster. We then repeatedly delete the vCluster Pod 100 times and check the micro-services application status in the member cluster using the kubectl command.

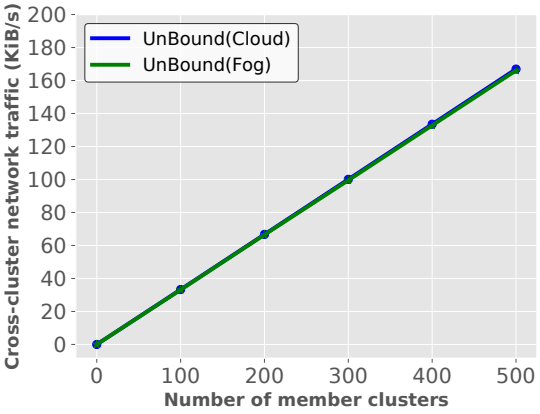
```
Command: kubectl get pods -n mgt-10-158-0-2-6443-klusterlet
NAME                                                      READY   STATUS    RESTARTS      AGE
adservice-7d857689bd-l9j7n-x-default-x-vcluster           1/1     Running   0             28m
cartservice-5d844fc8b7-2wt4n-x-default-x-vcluster         1/1     Running   0             28m
checkoutservice-84cb944764-22fx2-x-default-x-vcluster     1/1     Running   0             28m
coredns-f8bfc8497-cqnhb-x-kube-system-x-vcluster          1/1     Running   0             29m
currencyservice-76f9b766b4-f842p-x-default-x-vcluster     1/1     Running   0             28m
emailservice-767cd45966-nj8gm-x-default-x-vcluster        1/1     Running   0             28m
frontend-8475b5657d-kpsvd-x-default-x-vcluster            1/1     Running   0             28m
mgt-10-158-0-2-6443-klusterlet-76c679994b-7kn7z           1/1     Running   0             29m
mgt-10-158-0-2-6443-klusterlet-registration-agent-78c6ccdb7249w   1/1     Running   1 (28m ago)   29m
mgt-10-158-0-2-6443-klusterlet-work-agent-555896bddb-bdp2c        1/1     Running   0             28m
paymentservice-866fd4b98-p588p-x-default-x-vcluster       1/1     Running   0             28m
productcatalogservice-5b9df8d49b-nftmg-x-default-x-vcluster   1/1     Running   0             28m
recommendationservice-6ffb84bb94-t7bc4-x-default-x-vcluster   1/1     Running   0             28m
redis-cart-76b9545755-4zm2z-x-default-x-vcluster          1/1     Running   0             28m
shippingservice-648c56798-zpnmm-x-default-x-vcluster      1/1     Running   0             28m
vcluster-0                                                2/2     Running   0             8s
Command: kubectl get events -A --field-selector involvedObject.name=vcluster-0 --no-headers
 | grep Killing | grep Stopping.container.vcluster | wc -l
100
```

Figure 4.9 – Application stability despite a vCluster failure.

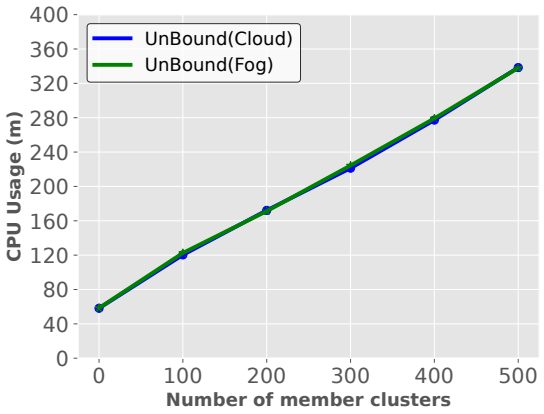
As shown in Figure 4.9, the first command lists all Pods running in the particular Namespace to check the information about these Pods, including their status, the times of restarts, and their age. The second command displays the number of times the vCluster has been deleted in the log files. We can see in the figure that after 100 deletions, the age of the vCluster Pod is only 8 seconds, while the other micro-service Pods (whose names end with vCluster) are 28 minutes, which shows that the vCluster is indeed deleted by our script. Note that the vCluster has 0 restart because we delete the Pod of the vCluster instead of restarting it. Additionally, the registration-agent restarts one time upon completion of the registration process. The figure shows that all micro-service Pods restarted 0 times, which indicates that the applications managed by the vCluster are not affected if the vCluster crashes. The micro-service application continues to serve users even after the main component fails. This result demonstrates the stability of UnBound.

4.4.4 One Management Cluster with Multiple Member Clusters

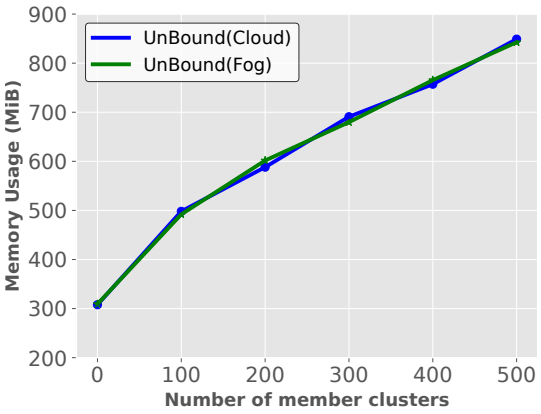
In this experiment, we explore the case of a single management cluster managing multiple member clusters. We deploy a management cluster with one VM that has 16 CPU cores and 32 GiB of memory. Then, we launch 100, 200, and up to 500 member clusters. Each member cluster uses only one VM with 2 CPU cores and 4 GiB of memory. We collect 2 hours of data for each round and average them to represent the results. We then run the experiment three times and present the average results of these three rounds



(a) Cross-cluster network traffic (collect in management cluster).



(b) CPU usage of API server in management cluster.

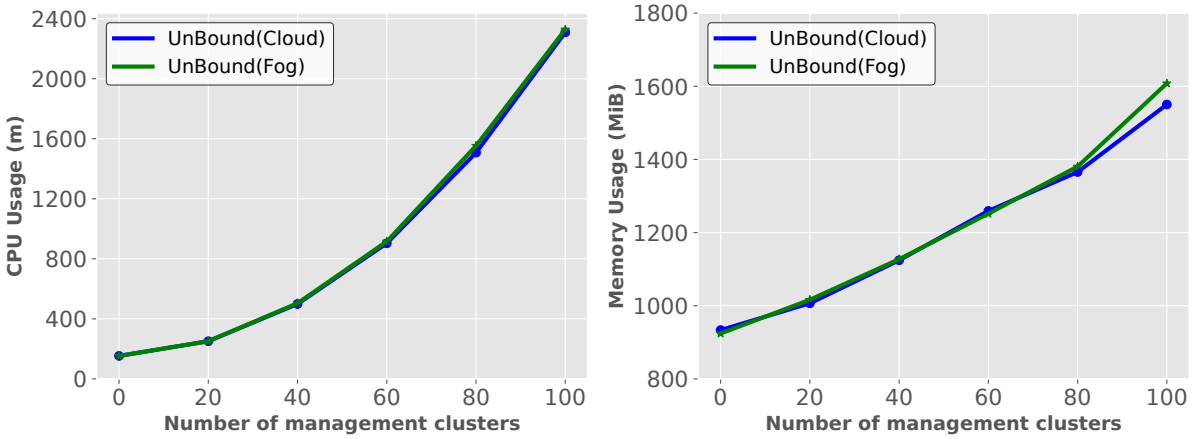


(c) Memory usage of API server in management cluster.

Figure 4.10 – Performance of UnBound with one management cluster managing multiple member clusters: Cross-cluster network traffic (a), and CPU and memory usage of the API server in the management cluster (b)(c).

in the figures. This experiment only shows the results of UnBound since the previous outcomes show a similar trend between our solution and Open Cluster Management.

We depict the results of cross-cluster network traffic between a management cluster and multiple member clusters in Figure 4.10(a). We find that the cross-cluster network traffic grows as the number of member clusters increases and is similar regardless of whether the environment is cloud or fog. In the case of 500 member clusters, the network traffic is around 166 KiB/sec in both cloud and fog cases. At the same time, Figure 4.10(b) and Figure 4.10(c) show the resource usage of the API server in the management cluster since the work-agent in member clusters will send requests to the API server of its management



(a) CPU usage of API server in member cluster. (b) Memory usage of API server in member cluster.

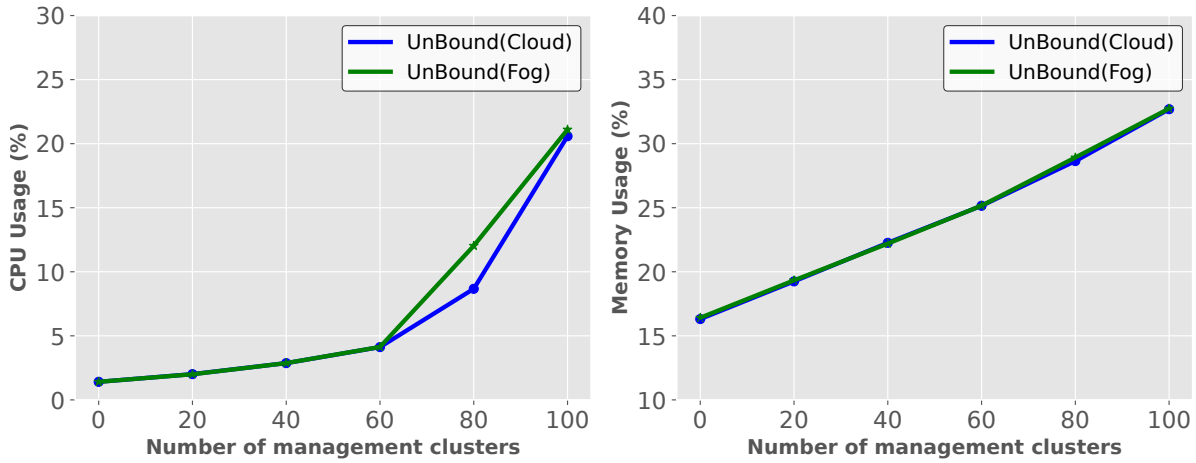
Figure 4.11 – Performance of API server in the member cluster while UnBound with multiple management clusters managing the same member cluster.

cluster to pull the latest prescriptions. The CPU and memory usage also follow the same trend as the results of network traffic, which rises linearly as the number of member clusters grows. The CPU usage in the fog environment is from 58 (0 member cluster), 123 (100 member clusters), to 338 m (500 member clusters), whereas the memory usage in the fog environment is from 309, 492, to 843 MiB.

4.4.5 Multiple Management Clusters with One Member Cluster

We now study the scenario of multiple management clusters managing a single member cluster. We launch a 101-VMs member cluster, consisting of one control plane node with 16 cores and 32 GiB of memory, and 100 worker nodes with 2 cores and 4 GiB of memory per worker. Next, we deploy 20, 40, and up to 100 management clusters, each with a single VM equipped with 2 CPU cores and 4 GiB of memory. Similar to the experiment in Section 4.4.4, we conduct three rounds of experiments; the outcome of each round is the average of 2 hours of data. The final results are the average of these three rounds.

Figure 4.11(a) and Figure 4.11(b) illustrate the CPU and memory consumption of the API server in the member cluster, respectively. In the cloud/fog environment, the CPU usage of the API server exhibits superlinear growth, rising from 153 m/153 m (for 0 management cluster) to 251 m/251 m (for 20 clusters) and reaching 2308 m/2328 m (for 100 clusters). In contrast, memory usage growth is relatively slow compared to CPU usage, demonstrating a linear trend from 934 MiB to 1550 MiB in the cloud scenario. The



(a) CPU utilization rate of member cluster.

(b) Memory utilization rate of member cluster.

Figure 4.12 – Performance of whole member cluster while UnBound with multiple management clusters managing the same member cluster.

growth in CPU usage results from the fact that agents of OCM in the member cluster send requests to the API server not only to the management cluster but also to the member cluster. Additionally, the components of vCluster also send requests to the API server. This surge in requests leads to increased processing time on the API server, potentially causing delays and timeouts that necessitate re-sending requests, further exacerbating the CPU usage. We however note that the absolute numbers remain reasonable, with a member cluster needing to allocate around 2 CPU cores and 1.5 GiB of memory when being a member of 100 federations simultaneously. One potential way to reduce these numbers could be to scale the number of API servers in the member cluster and enable load balancing to distribute requests across multiple API servers.

Figure 4.12(a) and Figure 4.12(b) depict the utilization percentage of the entire member cluster, including control plane and worker nodes. The CPU usage percentages exhibit relatively flat growth until the member cluster is shared between 60 management clusters. Subsequently, CPU usage increases sharply due to the high load on the API server in the control plane and Kube-proxy in each node. At the same time, the percentages of memory cluster utilization rise steadily from 16% to 33%.

We conclude that, although it is not totally negligible, the performance overhead of UnBound for both the management and the member clusters remains reasonable. This demonstrates the feasibility of realizing our vision of very large-scale fog meta-federations capable of spanning entire countries or even continents.

4.5 Conclusion

This chapter presents a new concept called meta-federations, which enables fog clusters to federate their resources with one another in a very flexible way, potentially allowing one to build very large-scale distributed fog platforms at the scale of a country or even a continent. We propose UnBound, a solution for meta-federations with scalability and different levels of multi-tenancy management in mind. UnBound leverages Kubernetes for resource orchestration within individual fog clusters. Then, we rely on Open Cluster Management and Virtual Kubernetes Clusters project as the main building blocks of the UnBound platform. Extensive experiments with actual large-scale deployments up to 500 *clusters* show that UnBound achieves inter-user and inter-federation isolation while maintaining performance comparable to the original Open Cluster Management and acceptable overhead levels.

Chapter 5

Efficient Monitoring Frameworks in Geo-Distributed Cluster Federations

5.1 Introduction

In a cluster federation, a "management cluster" is in charge of deciding which of the "member clusters" will be in charge of handling each newly deployed application. Although the original KubeFed project allowed little control of the choice of member cluster when applications should be deployed [120], newer federation frameworks such as mck8s support a range of fine-grained placement policies based on metrics such as cluster load, location, and network usage [31]. These policies base themselves on detailed monitoring information about the status of available resources in the respective target cluster, provided by a robust monitoring framework such as Prometheus and its extension Prometheus Federation [32], [33].

Monitoring a large cluster federation is a very challenging task because the number of metrics and the volume of monitoring data to be reported to the management cluster may grow to large values. To illustrate this problem, we leverage a real deployment in the Grid'5000 testbed [30]. In the setup, we use the "Kubernetes in Docker" (kind) framework to launch large numbers of Kubernetes clusters [184]. The first cluster acts as our management cluster. Then, we launch up to 500 member clusters. Each cluster contains two servers (one control plane and one worker node), resulting in up to 1,000 nodes in total.

Figure 5.1 depicts the aggregate volume of cross-cluster network traffic after deploying a large mck8s federation with no application workload. *recv* and *send* show the network traffic received/sent by the management cluster. We sum *recv* and *send* as the *total* network traffic. The scrape interval of Prometheus is set to 5 seconds, which means that the

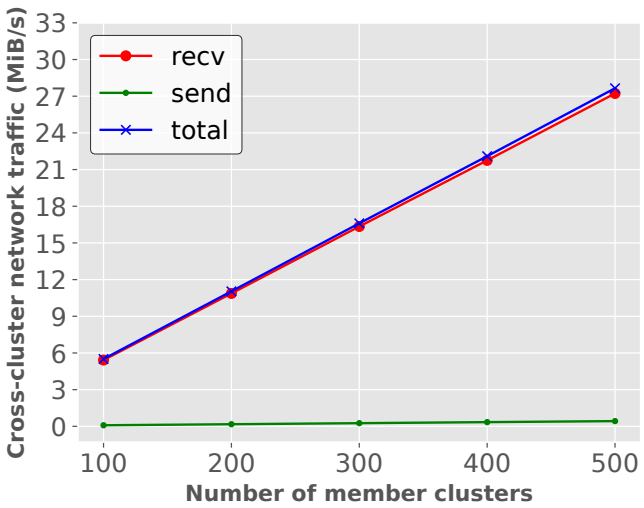


Figure 5.1 – Cross-cluster network traffic in the management cluster when using mck8s.

management cluster fetches metrics from every cluster once every 5 seconds. We observe a linear growth up to 27.7 MiB/s for monitoring 500 member clusters (1,000 nodes), which may be enough to saturate many fog computing networks and may eventually represent the majority of the system management traffic. The same linear growth appears when increasing the number of servers per cluster (not shown in the figure for clarity reasons). This very large management traffic is due to the resource monitoring used by mck8s to implement sophisticated scheduling functionalities. It does not appear when using KubeFed, which schedules workloads without considering the cluster status.

This experiment shows that even for medium-sized cluster federations, the necessary monitoring network traffic grows to such large values. This is caused by Prometheus Federation which offers fine-grained monitoring data that are being reported to the management cluster. Therefore, this chapter aims to reduce the volume of management data to provide the cluster federation with accurate enough and up-to-date information while significantly reducing the networking overhead of the federated monitoring framework itself. As a result, the precious platform's network resources may be used for actual user workloads rather than cluster management operations.

As discussed in Section 3.2.2, the Prometheus Federation has four limitations that make the system accurate but costly: (1) It scrapes monitoring data from every server in the federation; (2) It appends unnecessary labels to identify the monitoring data; (3) It fetches all monitoring data regardless of value changes; and (4) It collects the monitoring

data from target clusters at a fixed periodicity. To address these challenges, this chapter proposes Acala to address limitations (1), (2) and (3), and AdapPF to address limitation (4). Acala is an extension of Prometheus which uses two techniques to reduce the number of metrics to be reported to the management cluster: *metrics aggregation* merges together the metrics values with the same metric name and labels from multiple servers to report the aggregate status of an entire cluster rather than its individual servers; and *metrics deduplication* avoids one to repeatedly report the same metrics in case their value does not change. Our evaluations based on actual deployments in the geo-distributed Grid'5000 testbed [30] show that Acala reduces the volume of cross-cluster network traffic by up to 97% compared with vanilla Prometheus while reducing the necessary time to scrape metrics by up to 55% in a single member cluster experiment. At larger scales, Acala also performs well in reducing the cross-cluster network traffic by about 95%. The resource usage of Acala components also remains acceptable in the single cluster case, and we prove that our solution can save memory resources in the larger case. Moreover, a comparison of scheduling efficiency with and without data aggregation shows that aggregation has minimal effects on the system's scheduling function. We discuss Acala in detail in Section 5.2.

In Section 5.3, we present 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 to increase the frequency of data collection. This allows for the timely acquisition of up-to-date monitoring data, enabling the scheduler or system alarm to 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%.

We believe that the concepts proposed in Acala and AdapPF can be in principle integrated theoretically and practically together to enhance system performance and efficiency, considering that they address disjoint components of the Prometheus Federation architecture. However, due to the lack of time, we leave this topic for future work.

Parts of this chapter were published in [185]–[187].

5.2 Acala: Aggregate Monitoring for Geo-Distributed Cluster Federations

In this section, we first discuss the system design of the Acala framework and then turn to its performance evaluation.

5.2.1 System Design

The objective of Acala is to monitor computing resources in geo-distributed Kubernetes cluster federations while reducing the required cross-cluster network traffic as well as the deployment and configuration costs. In this section, we discuss the operation of Acala and introduce two data reduction strategies designed for Acala to reach our goal.

5.2.1.1 System Model

A geo-distributed Kubernetes cluster federation is a set of multiple "member" Kubernetes clusters in various locations that are considered as a single execution platform thanks to a "management" cluster which is in charge of collecting metrics data from the member clusters and deciding which application should be running in which member cluster. Each cluster consists of several computing nodes, and we assume that each node has enough resources to run the necessary applications to provide monitoring. All computing nodes in a cluster are located in the same area. The network connects each node and cluster and can communicate. Although the current design can support multiple layers, for the sake of simplicity, we leverage a two-tier architecture in this chapter.

Acala is built on several components from the Prometheus ecosystem, including the Prometheus server, Node-exporter [122], and Pushgateway [188]. The system overview is shown in Figure 5.2.

Prometheus server in member clusters: The duty of these servers is to scrape time-series data about local metrics in each member cluster and store them in their local database. They constitute the source of data before aggregation. They can also be used for querying detailed per-node metrics, for example for anomaly detection, diagnosis, or system management purposes. Moreover, these Prometheus servers can also be configured to trigger alerts about nodes with abnormal metric values in their member cluster, such as fully saturated nodes.

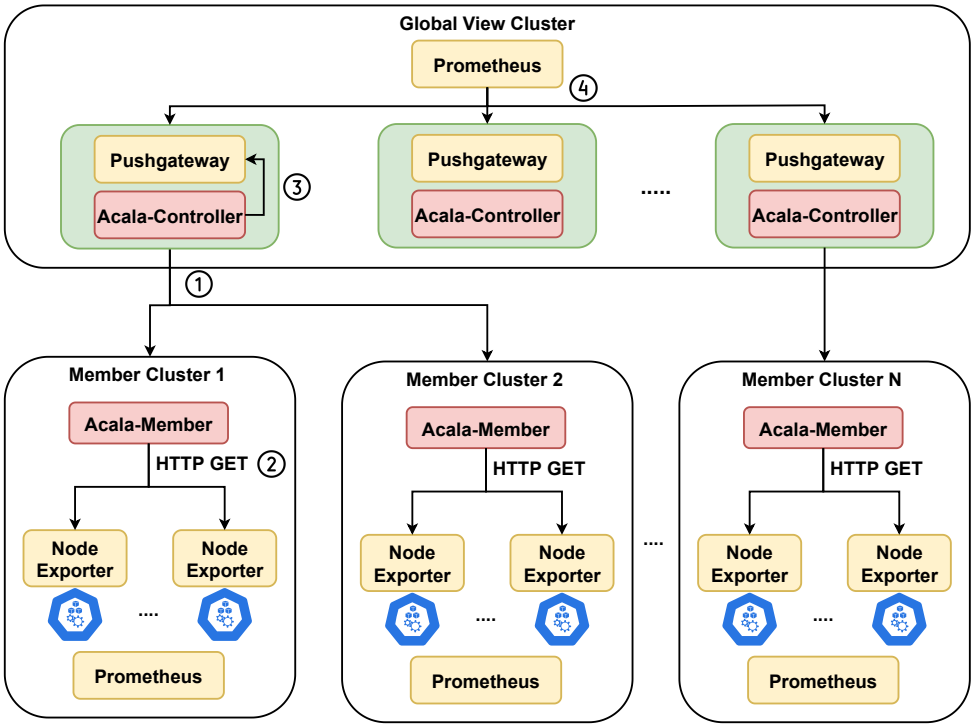


Figure 5.2 – Overview of Acala architecture and scrape flow.

Prometheus server in the global view cluster: The Prometheus server in the global view cluster is used to save the aggregated data from the member clusters and their local metrics. The federation's scheduler can leverage this Prometheus server to query member cluster information and make the scheduling decisions.

Node-exporter: This component monitors the per-node metrics. We install a Node-exporter for each node in each cluster to expose hardware and operating system metrics.

Pushgateway: The Pushgateway is installed in the global view cluster. It is a middleware that can expose these metrics for the Prometheus server to scrape. Moreover, Pushgateway also acts as a cache for metrics values.

Acala introduces two new components: Acala-Controller and Acala-Member. Acala-Controller is responsible for scraping the metrics from the target member cluster, adding the labels to identify the member cluster, and pushing the metrics to the Pushgateway. Acala-Controller is located in the global view cluster, and the administrators may launch additional Acala-Controller instances to accommodate the larger number of member clusters. In this case, each Acala-Controller can be configured to scrape metrics from a designated subset of member clusters. The task of Acala-Member is to pull the metrics

from the Node-exporter in a single member cluster and execute proposed data reduction strategies. The data transmissions between Acala-Controller and Acala-Member are compressed using gzip. The detailed scrape steps are as follows:

(1) When it is time for the Acala-Controller to scrape the metrics, the controller sends a request to the target Acala-Member.

(2) After Acala-Member receives the request, Acala-Member uses the HTTP GET method to pull the metrics from the local computing nodes through the Node-exporter. Meanwhile, Acala-Member executes Algorithm 1 to modify the metrics. Finally, Acala-Member compresses the metrics and sends them back to Acala-Controller.

(3) Acala-Controller decompresses the metrics and leverages the HTTP POST method to push metrics to the Pushgateway. In this step, the Acala-Controller adds the labels (IP address of control plane and cluster name) to identify the member cluster.

(4) The global-view Prometheus server periodically scrapes the metrics from the Pushgateway (at a user-defined periodicity independent from the periodicity of cross-cluster metrics transfer) and stores them locally. The administrator or federation scheduler can then query the monitoring data of the member cluster via this Prometheus server.

5.2.1.2 Timing to Scrape Metrics

Similar to the original design of Prometheus, the timing to scrape the metrics from the target member cluster is determined by a fixed scrape interval. We leverage a timer in the Acala-Controller to perform periodic scrape actions. When the timer counts down to 0, the system scrapes the metrics once and then sets the timer back to the default values configured by the administrator. A shorter scrape interval value means that data in the global view cluster will be more precise in representing the actual status of the member clusters yet at the cost of additional cross-cluster network traffic. The default scrape interval is defined as 5 seconds.

5.2.1.3 Data Reduction Strategies

To address the problems mentioned in Section 3.2.2, we propose two data reduction strategies: metrics aggregation and metrics deduplication highlighted in Algorithm 1.

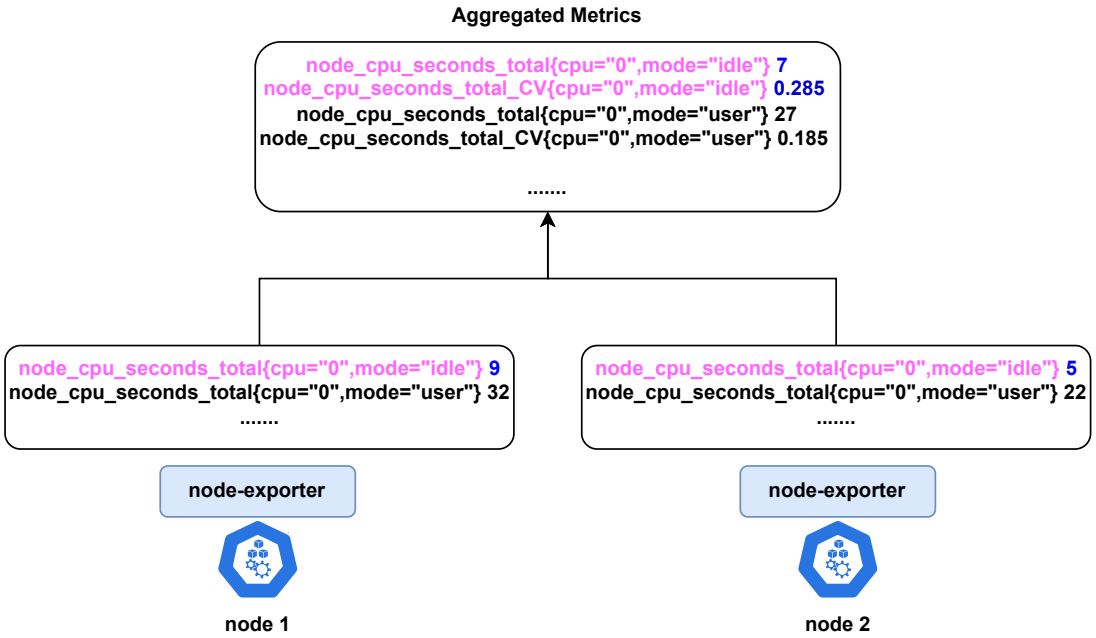


Figure 5.3 – An example of metrics aggregation.

The data model of metrics in Prometheus is composed of a *metric_name*, any number of pairs *label_name*, *label_value*, and finally a *metric_value*. The notation of a metric is:

$$metric\_name\ \{label\_name = label\_value,\ \ldots\}\ metric\_value$$

Metrics Aggregation. Each node in member clusters deploys the Node-exporter to expose its node-related metrics. In standard Prometheus Federation design, the highest scrape level is *job*, which will scrape metrics that are all nodes in the target member cluster and append all original labels for these metrics. In contrast, we choose metrics aggregation between the nodes in the target member cluster as our solution, elevating the point of monitoring view from "node" to "cluster." For easy understanding, we use *metric name with labels* to represent *metric name*, *label name*, and *label value*.

Figure 5.3 presents an example of metrics aggregation. Node-exporter of node 1 exposes the metric *node_cpu_second* $\{cpu = \text{“0”}, mode = \text{“}idle\text{”}\}$ 9, and node 2 has the same metric name with labels (fuchsia color). Metrics aggregation will thus aggregate both metric names with their labels and metric values (blue color). The resulting aggregated metric values are composed of the average of all individual values as well as the coefficient

of variation between them[1]. Note that the federation schedulers only need to know about the general status of the member clusters rather than detailed per-server metrics.

Node-exporter exposes node-related metrics such as utilized CPU, memory, and network bandwidth. These metrics can be aggregated with other metrics with the same name and labels. When metrics do not have identical name and labels within the cluster, Acala reports them non-aggregated to the global view cluster. In case more detailed per-node information is needed, the administrator can request the Prometheus server deployed in each cluster directly.

The main idea of this strategy is to aggregate values whose metric name and labels are identical in different servers. This method can collect and report monitoring data from each node in the target member cluster while significantly reducing cross-cluster network traffic. Moreover, metrics aggregation averages metrics values to represent the overall cluster status. This is similar to other related work [31], which also applies the aggregating strategy to represent the overall cluster resources situation. However, they perform aggregation *after* all individual metrics have been scraped, transferred, and stored in the global view cluster.

Prometheus Federation adds all original labels in each metric to identify which server each metric belongs to. In contrast, metrics aggregation keeps the metrics labels unchanged, the same as before aggregation. For the cluster information, we add the labels including the IP address of the control plane and cluster name (set by administrators manually) to indicate the member cluster in Acala-Controller, which takes place after the transmission. Therefore, metrics aggregation can reduce more cross-cluster network traffic.

Metrics Deduplication. Prometheus Federation blindly scrapes metrics from member clusters at a periodic interval. As a result, in case some metrics values do not change frequently, they get transferred repeatedly and unnecessarily, which consumes network bandwidth to transfer these redundant data. To further reduce cross-cluster network traffic, we propose a second data reduction strategy – *metrics deduplication*.

Metrics deduplication compares each aggregated metric value with the most recently transferred one. If the value is identical, the deduplication strategy removes this metric from this metrics transfer. On the other hand, if the metric value changes, the system will include this metric again to report the fresh data.

1. In statistics, the coefficient of variation is a standardized measure of the dispersion of aggregated values. It is defined as the ratio between the standard deviation σ and the mean μ of the distribution: $CV = \frac{\sigma}{\mu}$.

However, note that Prometheus includes a metrics staleness mechanism. If no new value is reported after 5 minutes (default of Prometheus), this metric will be marked as stale, and its value will be excluded from results returned to the federation scheduler. When using metrics deduplication, this staleness mechanism may exclude valuable deduplicated values from the results. Therefore, Acala leverages Pushgateway to cache these metrics locally so that the Prometheus server in the global view cluster can scrape from Pushgateway and keep fresh metrics values in the Prometheus server without having to repeatedly transfer them from member clusters.

To allow Acala to perform both metrics aggregation and deduplication, the algorithm will perform aggregation first and then deduplication based on the aggregated data. Although both data reduction strategies may run independently, we leave this topic for future work.

The metrics aggregation and deduplication process are illustrated in Algorithm 1. When a request for a new scrape action arrives at the Acala-Member in the target member cluster, the Acala-Member checks the type of request. If it is a full request, the algorithm clears the *LastAverage* and *LastCV* which contains the latest reported metrics values (lines 32-33). Then, Acala-Member pulls the metrics from each node through the Node-exporter. If the metric name with labels is already present in Aggregated Metrics AM, the value of matched metrics is appended to it. However, if AM does not have the same metric name with labels, the algorithm adds it as a new metric (lines 1-11). After all metrics finish aggregation, the algorithm computes the average and coefficient of variations of each metric (line 35). If deduplication is enabled, the function then checks *LastAverage* and *LastCV*. If *LastAverage* and *LastCV* exist, it means that the computed metric values should be compared with the previous one. If the values stored in *LastAverage* or *LastCV* compare to the current values are not identical, the algorithm appends the new value to the deduplicated $AMWCV$ file. If deduplication is disabled and/or *LastAverage* and *LastCV* are empty, then the system creates a full $AMWCV$ file (lines 17-27). After returning this file, the procedure copies the current values to *LastAverage* and *LastCV* (line 38). Finally, Acala-Member compresses the $AMWCV$ file using gzip, sends it back to Acala-Controller, clears the data, and waits for the subsequent scrape request (lines 41-43).

Metrics aggregation and metrics deduplication are well-established techniques. However, our work applies these methods and implementation within a geo-distributed Kubernetes cluster federation environment to build a fog computing platform where this

Algorithm 1: metrics aggregation and deduplication

Output: $AMWCV$: A File of Aggregated Metrics With Coefficient of Variation

```
1  Function Aggregation():
2      M_node ← Pull Metrics from each node in the cluster
3      if AM == ∅ then
4          AM ← M_node
5      else
6          for key, value ∈ M_node do
7              if key ∈ AM then
8                  AM_key.append(value)
9              else
10                 AM_key ← value
11     return AM
12 Function Calculation(AM):
13     for key ∈ AM do
14         AverageDict_key ← MEAN(AM_key)
15         CVDict_key ← STD(AM_key)/AverageDict_key
16     return AverageDict, CVDict
17 Function Dedup(AM, AverageDict, CVDict, LastAverage, LastCV, DedupFunc):
18     for key ∈ AM do
19         if DedupFunc then
20             if LastAverage and LastCV then
21                 if LastAverage_key != AverageDict_key or LastCV_key != CVDict_key then
22                     Build AMWCV based on AverageDict and CVDict (Deduplicated)
23             else
24                 Build AMWCV based on AverageDict and CVDict (Full)
25         else
26             Build AMWCV based on AverageDict and CVDict (Full)
27     return AMWCV
28 Function Main:
29     while true do
30         Wait for the connection
31         if Received scraping request then
32             if It is a full request then
33                 clear LastAverage and LastCV
34             AM ← Aggregation()
35             AverageDict, CVDict ← Calculation(AM)
36             if deduplicationfunction is enabled then
37                 AMWCV ← Dedup(AM, AverageDict, CVDict, LastAverage, LastCV, 1)
38                 LastAverage ← AverageDict, LastCV ← CVDict
39             else
40                 AMWCV ← Dedup(AM, AverageDict, CVDict, LastAverage, LastCV, 0)
41             Compress AMWCV
42             send AMWCV back to Acala-Controller
43             AM, AverageDict, CVDict ← ∅
```

environment has yet to be extensively explored. Moreover, the proposed framework and two data reduction strategies elevate the traditional view of monitoring in Prometheus Federation from "node" level to "cluster" level. The design of Acala is to hierarchically monitor different levels of metrics. The original Prometheus Federation scrapes per-server metrics from all member clusters to the global view cluster, where all the detailed metrics can be found. Instead, Acala keeps the detailed per-server metrics in the member cluster, which are neither aggregated nor deduplicated. It then reports the modified metrics to the global view cluster. Using metrics aggregation, the monitoring data in the global view cluster represents the overall member cluster status. The layer of monitoring will be "cluster status" in the global view cluster and "node status" in each member cluster. Note that, although Acala performs metrics aggregation and metrics deduplication, from a macro perspective, our solution does not discard any data. The operator can still query detailed per-node metrics in member clusters for anomaly detection and system management.

Prometheus also supports a feature called "recording rules" which is similar to Acala's metrics aggregation. Using it, one can pre-aggregate selected metrics, store the results in member clusters, and scrape them from other Prometheus servers with appropriate labels. However, recording rules in Prometheus need to be defined manually for each metric in each member cluster, which is error-prone and may increase the deployment and configuration cost in large-scale environments. Moreover, Prometheus does not provide metrics deduplication, so it reports data to the global view cluster periodically, regardless of whether the value has changed since the previous scraping period.

5.2.1.4 Kubernetes Deployment and Error Handling

Acala is designed to use Kubernetes to manage its own deployment due to Kubernetes "graduated" maturity level certified by the Cloud Native Computing Foundation (CNCF) [24]. This maturity level is usually considered a stable and production-ready solution. Using this level of container orchestrator can make the design of Acala closer to the real environment and further improve the current environment. However, we argue that the concepts and algorithms proposed in this chapter can be easily applied and integrated with other current or future monitoring solutions and container orchestrators.

Kubernetes offers various workload resource types such as Deployment and Job [189]. We leverage a Kubernetes Deployment for deploying the Pushgateway, Acala-Controller,

and Acala-Member in their respective Kubernetes cluster. A critical advantage is that these applications will automatically restart if any component failure occurs.

The activation of deduplication requires a careful system design to ensure that metrics values are not lost. Acala-Controller uses HTTP POST to send metrics to the Pushgateway. If the Acala-Controller fails to send data to the Pushgateway, it assumes that the Pushgateway may have failed and lost previous metrics values. It therefore continues to scrape the metrics periodically from Acala-Member, but it sends full data requests. When the Pushgateway recovers, the Acala-controller can give it a fresh set of metrics values before returning to its normal behavior.

If an error occurs with the Acala-Controller and Kubernetes decides to restart it, the Acala-Controller will behave as if it was its first time launch. It then sends a first full data request to the Acala-Member before starting again to accept deduplicated metrics values.

Finally, in case Acala-Member fails, it will restart with an empty $LastAverage$ and $LastCV$, and thus send the full data to the Acala-Controller before returning to its normal behavior.

5.2.2 Performance Evaluation

We evaluate Acala's performance using four separate experiments: (i) replaying Google cluster-usage traces in a member cluster to study the distribution of monitored metrics values across the cluster's servers; (ii) evaluating the cross-cluster network traffic and other performance indicators for a single member cluster with the various number of worker nodes; (iii) exploring Acala's scalability with a greater number of member clusters; and (iv) scheduling workloads across member clusters with Acala monitoring framework.

5.2.2.1 Experimental Setup

For the sake of making our work as close as possible to a production environment, we implement a prototype of our framework and run it in the Grid'5000 geo-distributed testbed [30]. We discuss the setup along the following four aspects: deployment of the experiment, performance indicators, comparison methods, and tools for collecting the data.

Deployment. To support the design features described in Section 5.2.1, we utilize Python 3.10 to implement Acala-Controller and Acala-Member. We leverage Kubernetes (v1.23.5) for container orchestration to build the test environment and analyze Acala in a

geo-distributed cluster federation. At the same time, we use different open-source projects in Kubernetes clusters for different functions. Cilium v1.11.4 is our Container Network Interface (CNI) that provides, secures, and observes network connectivity between container workloads in Kubernetes. Kube-Prometheus-stack v34.10.0 is a collection of Kubernetes manifests, including Prometheus v2.34.0 and Node-exporter v1.3.1.

We launch one management (global view) cluster and one or more member clusters.

— The management cluster contains two nodes (one for the control plane and one worker node). Each node runs inside a VM with 4 CPU cores and 16 GiB of memory for all four experiments.

— In the first and second experiments, we create a single member cluster with a number of nodes between 2 and 31 nodes. The VMs in the member cluster have 2 CPU cores and 8 GiB of memory.

— In the third experiment, we launch up to 50 member clusters, each of which has 20 nodes (1,000 nodes in total). To mimic the limited resources of worker nodes in a fog computing environment, the VMs in the member clusters have 1 CPU core and 4 GiB of memory for worker nodes and 2 CPU cores and 8 GiB of memory for the control plane.

— In the fourth experiment, we run 5 member clusters, each of which contains 1 control plane and 6 worker nodes. Each VM in all member clusters is equipped with 2 CPU cores and 8 GiB of memory.

We deploy Acala-Controller in the global view cluster and Acala-Member in each member cluster. Acala-Controller is installed on the same node as the Prometheus server, which can reduce the inter-node network traffic when the Prometheus server in the global view cluster scrapes from Pushgateway. Meanwhile, the Pushgateway is launched in the same Pod as Acala-Controller, which enables local metrics transmission within this Pod.

Performance Indicators. The first experiment aims to evaluate the dispersion of metrics values across an active member cluster. We therefore evaluate the Coefficient of Variation (CV) across values of the same metrics among the member cluster. The main goal of Acala is to reduce the cross-cluster network traffic in geo-distributed cluster federations. Hence, in the second and third experiments, we measured network traffic as our primary indicator. Lower network traffic implies better performance. Moreover, efficiency is a pivotal point in evaluating a system. Therefore, the scrape duration and resource consumption are also the objectives we consider. The overall efficiency is better

if scrape duration and resource consumption are shorter and lower. The objective of the fourth experiment is to understand the impact of monitoring data accuracy with the Acala framework. Therefore, we inject and schedule the workloads across the member clusters based on the resource status of workloads. Then, we check how many tasks can be completed as the indicator. A higher completion rate means better performance.

Comparison Methods. In our proposed system, the data reduction strategy is a method to reduce the metrics when the global view clusters scrape from the member clusters. To evaluate the performance of metrics aggregation and metrics aggregation with deduplication, we compare them with unmodified Prometheus Federation. Comparing our framework to a production-ready monitoring solution as a baseline can better reflect the effectiveness of our approach in real-world scenarios. In addition, we examine these three methods with different scrape interval (5 s and 60 s).

Tools for Collecting the Data. The results of experiments in Section 5.2.2.3 and Section 5.2.2.4 are gathered for 6 minutes. Three performance indicators that need other tools or functions to collect related data for evaluating Acala. For the cross-cluster network traffic, we use *tcpdump* to capture the network traffic. The scrape duration is based on the *time.perf_counter()* function in the Acala source code to measure the execution time of each step. We sum the execution time of Acala-Member, Acala-Controller, and the duration of Prometheus scrape from the Pushgateway to become our scrape duration. The resource usage of the Acala components, including CPU and memory, is monitored by the Kubernetes Metrics Server (v0.6.1) [180].

5.2.2.2 Distribution of Resource Usage in One Member Cluster

Aggregating metrics values across a cluster obviously results in dropping detailed information about individual servers. We however argue that the resource usage of each worker node in a single Kubernetes cluster is sufficiently well balanced, so individual metrics values do not differ very much, and aggregate information is sufficient to perform accurate task scheduling. Note that, although Acala does not report per-server metrics to the management cluster, these measurements remain available in each member cluster (e.g., for troubleshooting).

To understand the distribution of resource usage in a member cluster, we replay a workload in the member cluster based on the *Google cluster-usage traces*, which is a real-world dataset from a Google cluster [132]. A Google cluster consists of multiple machines arranged in racks and interconnected through a high-bandwidth cluster network.

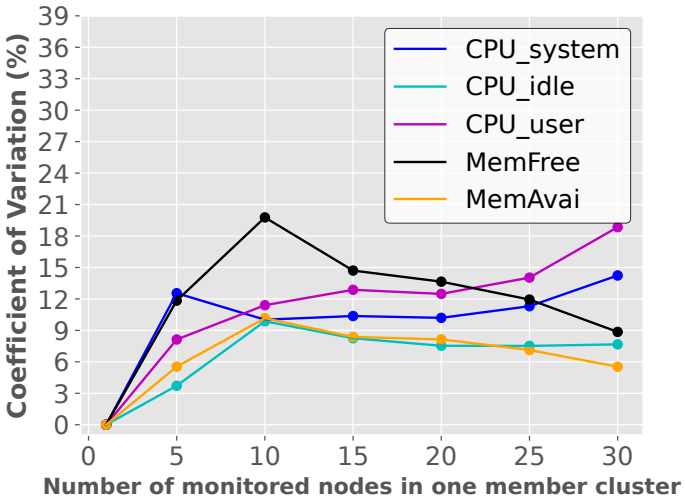


Figure 5.4 – Coefficient of variation when injecting workloads in a member cluster.

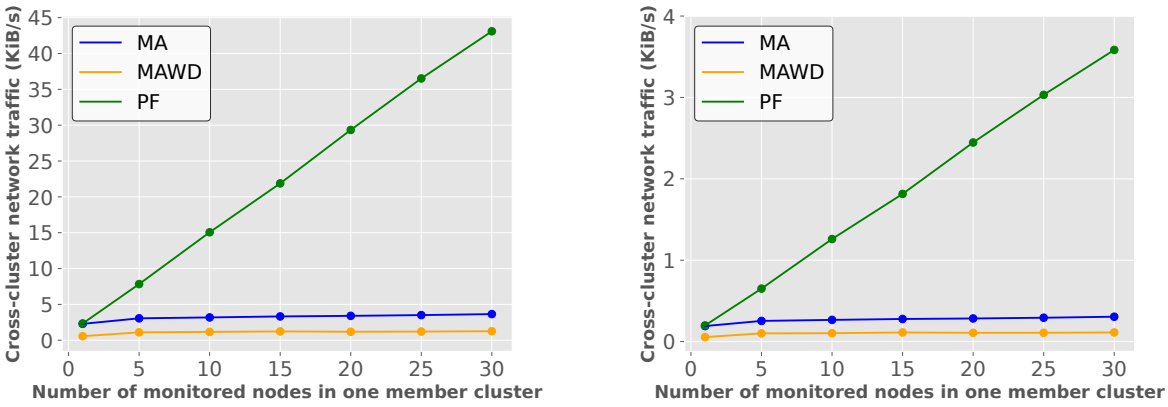
In this cluster, there is a shared cluster management system responsible for scheduling workloads to these machines in a cluster. The Google cluster-usage traces include data about thousands of deployed applications in a cluster with several important parameters, including resource requirements (CPU, RAM), duration, and inter-arrival rates. We build a container application based on the stress-ng tool [190] to generate actual resource usage. In each experiment, we inject the workload in the member cluster for a duration of 60 minutes (1,096 tasks in total) and then wait for 30 more minutes for letting jobs to complete and release the computing resources. We monitor five metrics in the cluster with a scrape interval of 5 s and compute the coefficient of variation across the cluster's servers for clusters configured with different numbers of servers. The metrics are chosen as follows:

— node_cpu_seconds_total{"mode=system"}: Time spent in kernel space of all the node's CPU cores. We use "CPU_system" in the figure.

— node_cpu_seconds_total{"mode=idle"}: Time during which each of the node's CPU cores remained idle. We use "CPU_idle" in the figure.

— node_cpu_seconds_total{"mode=user"}: Time spent in user space of all node's CPU cores. We use "CPU_user" in the figure.

— node_memory_MemFree_bytes: Free memory on the node. We use "MemFree" in the figure.



(a) Scrape interval set to 5 seconds.

(b) Scrape interval set to 60 seconds.

Figure 5.5 – Average cross-cluster network traffic with scrape interval set to 5 seconds (a) and 60 seconds (b).

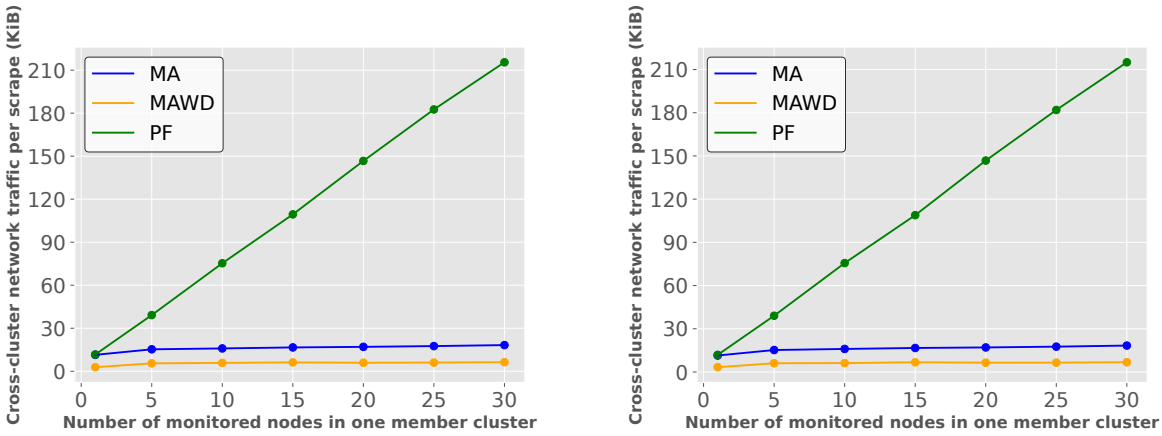
— node_memory_MemAvailable_bytes: Available [2] memory on the node. We use "MemAvai" in the figure.

The results of this experiment are plotted in Figure 5.4. We can see that the CV of all five metrics in most of the cases remains between 3.7% and 19.7% except for the single-node case where no inter-node variations exist and CV is therefore equal to 0. An interesting result is that the CV of CPU resources tends to grow for cluster sizes greater than 25 servers. The reason is that the workload can be handled by fewer than 25 servers, so some servers remain idle while others are active. This experiment shows that the distribution of resource usage metrics remains relatively well-balanced in a wide variety of situations, which indicates that Kubernetes does an excellent job at balancing the load across available servers. It also demonstrates that reporting only an aggregate of these metrics values to the management cluster depicts a sufficiently accurate picture of the cluster's situation to allow efficient scheduling decisions.

5.2.2.3 Performance in a Single Member Cluster

In this experiment, we evaluate the performance improvements brought by Acala's aggregation strategies using a single member cluster with a variable number of servers.

2. Available memory includes unallocated (free) memory as well as the cached and buffered memory that are currently occupied by the system but potentially reclaimable.



(a) Scrape interval set to 5 seconds.

(b) Scrape interval set to 60 seconds.

Figure 5.6 – Cross-cluster network traffic per scrape with scrape interval set to 5 seconds (a) and 60 seconds (b).

Cross-Cluster Network Traffic. Figure 5.5 and Figure 5.6 show the experimental results of cross-cluster network traffic on average and per scrape, respectively. Figure 5.5(a) and Figure 5.6(a) present the results of the system scraping the metrics every 5 seconds, whereas the outcomes of 60 seconds scrape interval are shown in Figure 5.5(b) and Figure 5.6(b). To increase the readability of the figures, we denote Metrics Aggregation as MA, Metrics Aggregation With Deduplication as MAWD, and Prometheus Federation as PF.

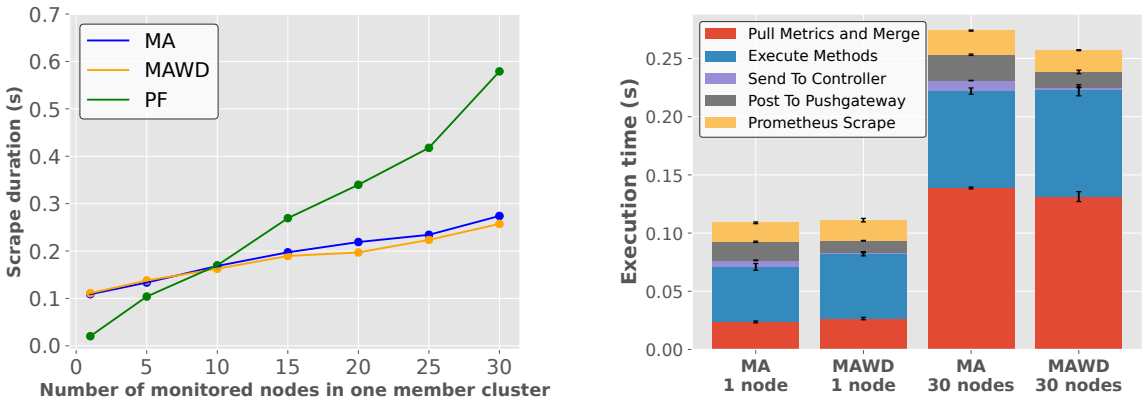
Figure 5.5(a) shows that metrics aggregation with deduplication significantly reduces cross-cluster network traffic, which is 0.56 KiB/s, whereas the network traffic in metrics aggregation and Prometheus Federation are 2.29 KiB/s and 2.33 KiB/s when monitoring a single worker node in the member cluster. Using metrics aggregation with deduplication in Acala and compared to Prometheus Federation, the reduction of network traffic in the single-node case is 1.77 KiB/s, which is 76% lower, and there are 2% lower when we apply the metrics aggregation as our data reduction strategy. Figure 5.5(b) shows the same trend that both of our proposed methods have lower network traffic than the Prometheus Federation. If the monitored nodes are set to 20, 25, and 30, the network traffic is 96%/88%, 97%/90%, and 97%/92% lower when we use the metrics aggregation with deduplication/metrics aggregation and compare to Prometheus Federation.

Overall, Figure 5.5 demonstrates that no matter how many monitored nodes are in the experiment, both of our proposed methods perform significantly better than Prometheus

Federation. The design of Prometheus Federation will scrape the metrics of all nodes in the target member cluster to the global view cluster. Our strategy is also to scrape the metrics that are all nodes, but we make this task in the member cluster, making the transmission happen inside the cluster, which can reduce the cross-cluster network traffic. Moreover, our methods aggregate the same metrics between the monitored nodes, which can decrease the volume of monitoring data to reduce cross-cluster network traffic and make the view of monitoring from node to cluster. In addition, the method of metrics aggregation with deduplication is even lower than metrics aggregation since unchanged data is not sent multiple times. If the value of the metric is the same as the current time and the last time, metrics aggregation with deduplication will remove these metrics to save network bandwidth between clusters.

Acala collects metrics from monitored targets based on a fixed scrape interval. However, the current design does not smooth the data transmission over time as data get transferred at periodic interval (same as Prometheus Federation). Therefore, we also want to know how much network bandwidth is used per scrape in this experiment. The results of cross-cluster network traffic per scrape are shown in Figure 5.6. In the case of 5 seconds scrape interval, we see in Figure 5.6(a) that the cross-cluster network traffic of Prometheus Federation experiences linear growth from 11.67 KiB (for 1 node), 39.17 KiB (for 5 nodes) to 215.48 KiB (for 30 nodes). The difference between 1 and 30 monitored nodes is 203.81 KiB, which is 1,746% greater. This is because Prometheus Federation scrapes the metrics that are all nodes in the member cluster. Moreover, it also appends all original labels in each metric to identify the scraped target. These strategies significantly increase cross-cluster network traffic. Figure 5.6(b) reflects that the results are almost the same as with 5 seconds scrape interval case in our methods of metrics aggregation and metrics aggregation with deduplication. The network traffic of both methods grows a little when the number of monitored nodes increases. When increasing the monitored nodes from 5 to 30, the network traffic of metrics aggregation with deduplication/metrics aggregation is 6.06/15.24 KiB and 6.74/18.33 KiB, respectively. The growth rates are around 11% and 20%, which are lower than the Prometheus Federation. Although our methods aggregate the metrics, some metrics are specific to nodes. These metrics will be appended to aggregated metrics, which will increase cross-cluster network traffic a little.

Scrape Duration. We now study the time it takes to scrape metrics using Acala. For the sake of clarity, we only show the results of 5 seconds scrape interval in Figure 5.7. We see in Figure 5.7(a) that scrape duration grows with the number of worker nodes that



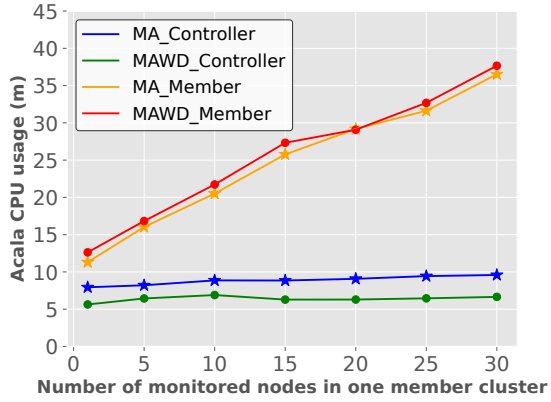
(a) Scrape duration.

(b) Execution time of each step.

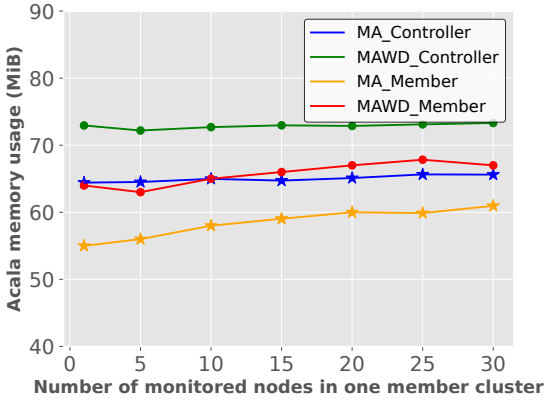
Figure 5.7 – Scrape duration (a) and execution time of each step (b) when scrape interval is set to 5 seconds.

need to be scraped. However, the growth rates of Prometheus Federation's scrape duration are greater than those of both of our methods. Acala starts to outperform Prometheus Federation with about 15 monitored nodes. In the case of a single node, the scrape duration of metrics aggregation and metrics aggregation with deduplication is greater than Prometheus Federation because Acala must execute additional operations compared to Prometheus Federation. In the case of 30 nodes in the member cluster, the scrape duration of Prometheus Federation is around 0.58 s, whereas the scrape duration of metrics aggregation is 0.27 s (53% lower than Prometheus Federation). The metrics aggregation with deduplication in the same case performs even better, up to 55% shorter than Prometheus Federation. In general, our methods perform better than Prometheus Federation when the cluster contains more nodes.

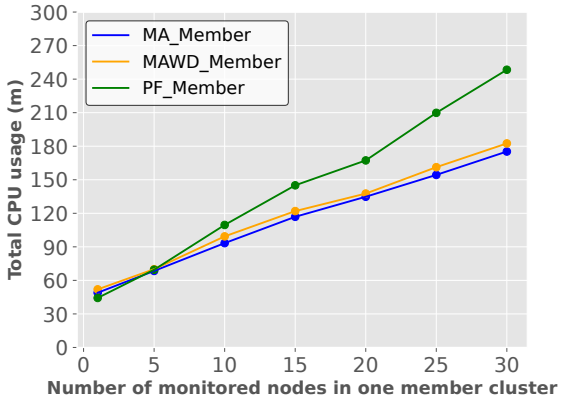
The detailed execution times of each step are shown in Figure 5.7(b). We present two cases with 1 node and 30 nodes and split the scrape time along the five main steps of Acala: *Pull Metrics and Merge*, *Execution Methods*, *Send To Controller*, *Post To Pushgateway*, and *Prometheus Scrape*. We can see that the total execution time of metrics aggregation and metrics aggregation with deduplication are similar in 1 node case. Based on the figure, we can find that the *Send to Controller* and *Post To Pushgateway* are slightly greater in both cases because metrics aggregation will not compare the last metrics values, which have more metrics that need to be sent and executed. The execution time of the 30 node situation is greater than 1 node. The major increases are from *Pull Metrics* and *Execution Methods*. More nodes need to be processed by the Acala-Member, which takes more time.



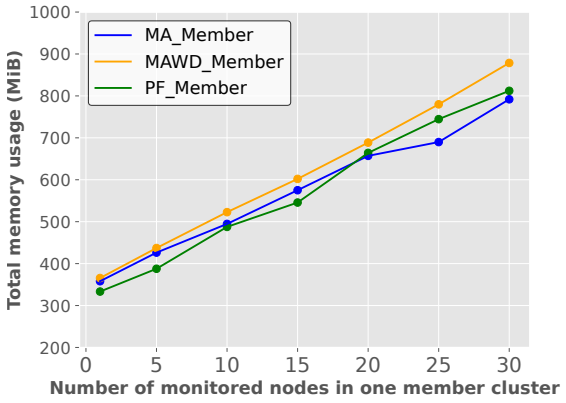
(a) CPU consumption of Acala components.



(b) Memory consumption of Acala components.



(c) Total CPU consumption of member cluster.



(d) Total memory consumption of member cluster.

Figure 5.8 – CPU (a) and memory (b) consumption of Acala components and total CPU (c) and memory usage (d) of member cluster when scrape interval is set to 5 seconds.

Resources Consumption of Acala Components and Member Cluster. To better understand the efficiency of our system, we measure the resource usage to see how much CPU and memory are needed. Same as scrape duration experiments, we only show the results of 5 seconds scrape interval in Figure 5.8. The CPU usage of Acala components is depicted in Figure 5.8(a). We found that the CPU usage of Acala-Member grows as the number of monitored nodes increases, and metrics aggregation with deduplication is a little greater than metrics aggregation. There are two reasons for these results: one is that more nodes need to execute, and the other is because comparison consumes CPU resources. At the same time, the Acala-Controller's CPU usage of metrics aggregation with deduplication is lower than metrics aggregation because the transmission volume

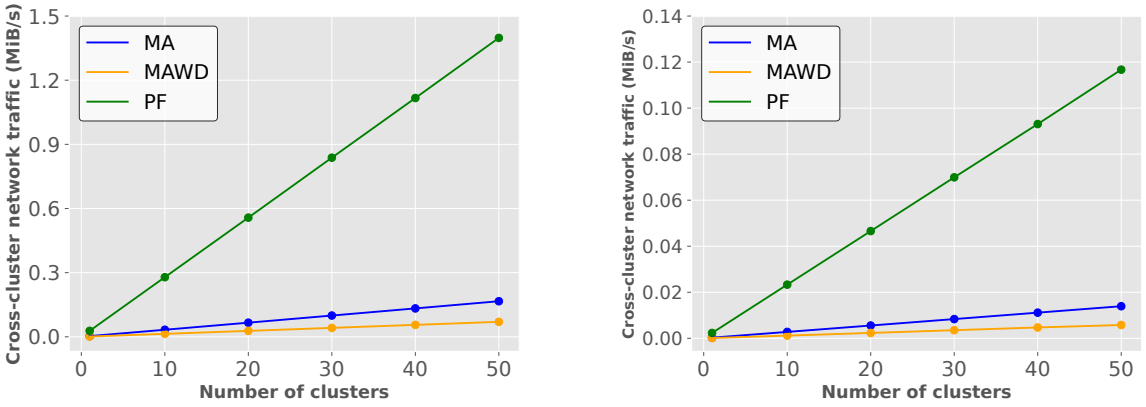
is smaller, which reduces the execution of functions such as decompression in the Acala-Controller. Regardless of the Acala-Controller or Acala-Member, the memory consumption of both components is under 80 MiB, as shown in Figure 5.8(b).

The total resource usage combines Acala-Member and Prometheus server resource consumption in a member cluster. In Figure 5.8(c), we can see that the CPU usage of Prometheus Federation is higher than both of our approaches when the number of worker nodes is greater. Although our methods require resources to run the data reduction strategies, Prometheus Federation needs to attach local labels to metrics, which also consumes resources. This is the reason why Prometheus Federation uses higher CPU resources. Figure 5.8(d) shows the results for memory. Overall, we do not see much difference in memory consumption between these three methods. For example, in the case of 30 computing nodes, the memory usage is 792 MiB, 879 MiB, and 812 MiB for metrics aggregation, metrics aggregation with deduplication, and Prometheus Federation, respectively.

5.2.2.4 Performance in Multiple Member Clusters

We now evaluate Acala in a larger environment where we increase the number of member clusters up to 50 clusters with 20 computing nodes each, representing up to 1,000 computing nodes. We present the same cross-cluster network traffic and resource consumption measures as in the previous section but exclude the scrape duration performance because it is a local measure within the cluster and would therefore show the same results as with a single cluster.

Cross-Cluster Network Traffic. Figure 5.9 and Figure 5.10 show the average and per-scrape network traffic when the number of clusters increases. In these two figures, subfigures (a) and (b) are the results of 5 seconds and 60 seconds scrape interval. Figure 5.9(a) presents the same trend as the previous experiment: Prometheus Federation still exhibits the greatest cross-cluster network traffic between the member and global view clusters. With 50 member clusters, Prometheus Federation uses around 1.40 MiB/s, followed by metrics aggregation (0.17 MiB/s) and metrics aggregation with deduplication (0.07 MiB/s). This means that metrics aggregation reduces the cross-cluster traffic by 88% compared with Prometheus Federation, and metrics aggregation with deduplication reduces it by 95%. Figure 5.9(b) obtains a similar reduction when scraping metrics at the 60 seconds interval, showing that Acala can effectively reduce the cross-cluster network bandwidth usage and free these precious resources to be rather used by actual user workloads.



(a) 5 seconds scrape interval.

(b) 60 seconds scrape interval.

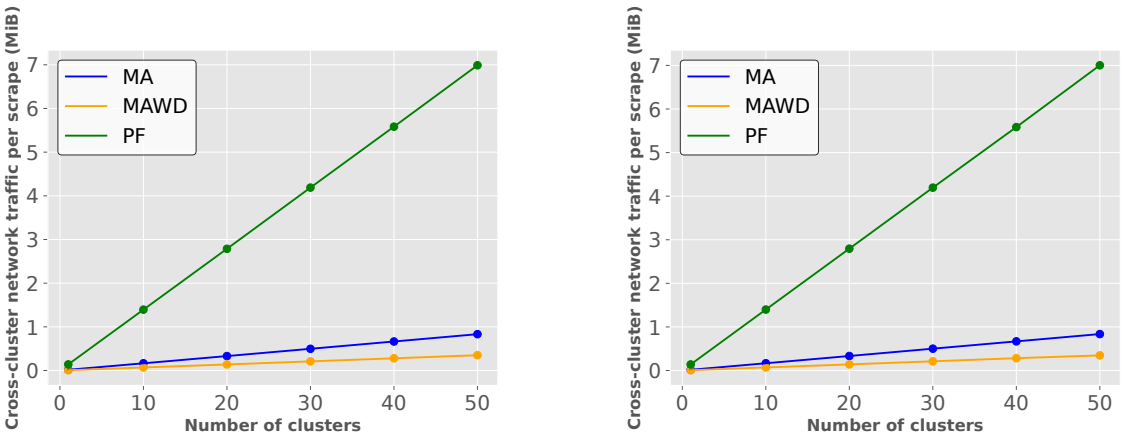
Figure 5.9 – Average cross-cluster network traffic in multi-cluster deployment with scrape interval set to 5 seconds (a) and 60 seconds (b).

Figure 5.10 shows that no matter whether the scrape interval is set to 5 or 60 seconds, the cross-cluster network traffic per scrape is almost identical. We see in Figure 5.10(a) that when the scrape interval is set to 5 seconds, cross-cluster network traffic of Prometheus Federation is 4.19, 5.58, and 6.99 MiB per scrape when using 30, 40, and 50 member clusters, which represent 600, 800, and 1,000 computing nodes in total. In the same conditions, the network traffic using metrics aggregation/metrics aggregation with deduplication is 0.50/0.21, 0.66/0.28, and 0.83/0.35 MiB. Comparing metrics aggregation and metrics aggregation with deduplication to Prometheus Federation, the reductions of network traffic are 3.69/3.98, 4.92/5.30, and 6.16/6.64 seconds which is around 90% lower than Prometheus Federation. Figure 5.10(b) shows almost identical results using a scrape interval of 60 s.

Overall, Figure 5.9 and Figure 5.10 show that our methods effectively reduce cross-cluster network traffic by an order of magnitude compared to the Prometheus Federation.

Resource Usage in the Management Cluster. We now explore the resource usage in the management cluster with a large number of member clusters. We only show the results of total usage in the management cluster with a 5 s scrape interval. As previously discussed, the total resource usage is summed by Acala components and the Prometheus server.

Figure 5.11(a) plots the CPU resource usage as a function of the number of member clusters. All approaches see a roughly linear growth of their CPU utilization. Also,



(a) 5 seconds scrape interval.

(b) 60 seconds scrape interval.

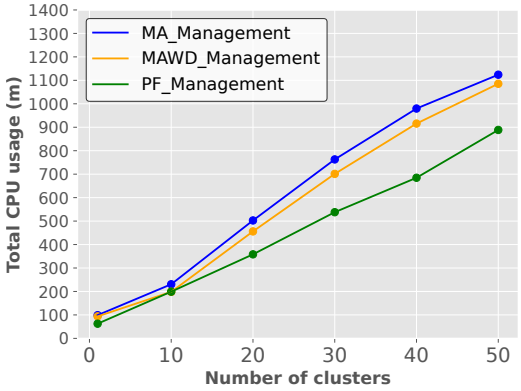
Figure 5.10 – Cross-cluster network traffic per scrape in multi-cluster deployment with scrape interval set to 5 seconds (a) and 60 seconds (b).

Acala's strategies require slightly more CPU than Prometheus Federation. The reason is that Acala needs to apply additional operations compared to Prometheus Federation: after scraping the metrics from member clusters, Acala-Controller then leverages HTTP POST methods to put these metrics in the Pushgateway. The metrics aggregation without deduplication is slightly higher than the metrics aggregation with deduplication because more metrics have to be scrapped.

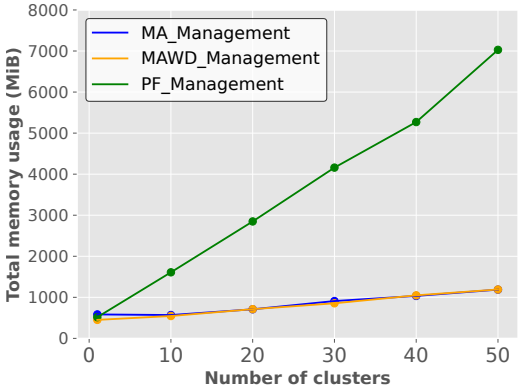
In contrast with CPU, Acala's memory usage is much lower than that of Prometheus Federation (Figure 5.11(b)). With 50 member clusters, the memory usage in the management cluster is respectively 7,028 MiB, 1,191 MiB, and 1,187 MiB for Prometheus Federation, metrics aggregation with deduplication, and metrics aggregation. The memory reduction in both methods compared to Prometheus Federation is 5,837 MiB (83% lower) and 5,841 MiB (83% lower) for metrics aggregation with deduplication and metrics aggregation.

5.2.2.5 Impact of Aggregation on Scheduling Efficiency

To understand the impact of aggregating monitoring data with the Acala framework, we compare scheduling efficiency based on monitoring data with and without aggregation. We inject the same workloads from Google cluster-usage traces as in Section 5.2.2.2 and simulate each task execution using stress-ng, with 60 minutes of injection and 30 minutes waiting for tasks to be finished. We set the scrape interval to 5 seconds for Prometheus



(a) Total CPU consumption of management cluster.



(b) Total memory consumption of management cluster.

Figure 5.11 – Total CPU (a) and memory usage (b) of management cluster when scrape interval is set to 5 seconds.

Federation and Acala. To schedule the workloads across these member clusters, we deploy mck8s to federate and manage clusters and workloads. We also modify the scheduler of mck8s to make it suitable for our experiment. The scheduler assigns each task to the cluster where the highest (worst-fit) or lowest (best-fit) number of Pods can be executed. Note that Kubernetes scheduling takes into account the sum of resource requests from running pods in each cluster node rather than their actual current resource usage. We then measure how many tasks can be completed as the metric of scheduling efficiency. We run each experiment 10 times and average the results across them.

Figure 5.12 depicts the completion rate when we use monitoring data for scheduling from Prometheus Federation and Acala with metrics aggregation. The two bars on the left represent the worst-fit, and the remaining are the best-fit method. The completion rate of using Prometheus Federation monitoring data is slightly higher than Acala in the worst-fit case, which are about 80.3% (Prometheus Federation) and 76.2% (Acala). On the other hand, the best-fit method selects the member cluster with the smallest available resources to enhance the resource utilization of clusters, which requires greater accuracy in monitoring data to schedule to the correct member cluster. In this difficult case, the results are similar to worst-fit, which are 79.8% (Prometheus Federation) and 75.7% (Acala). Although metrics aggregation slightly impacts the data accuracy, this scheduling experiment and previous results show that our solution can reduce cross-cluster

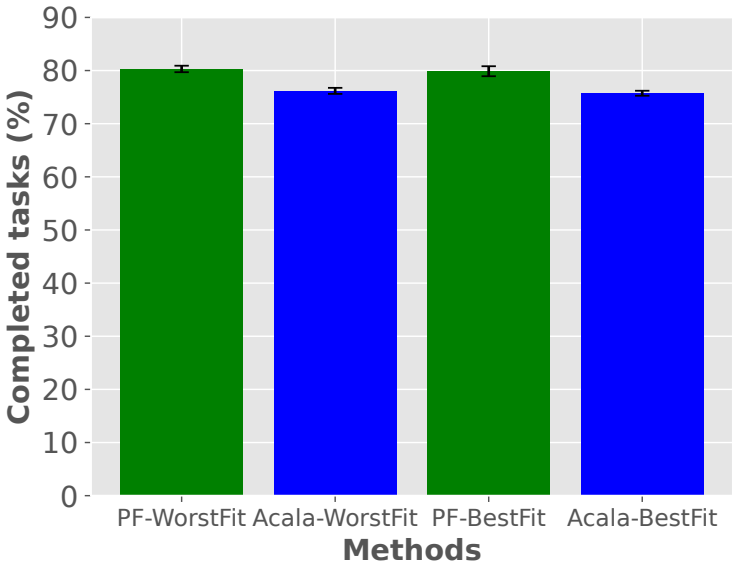


Figure 5.12 – Completion rate when injecting workloads in 5 member clusters.

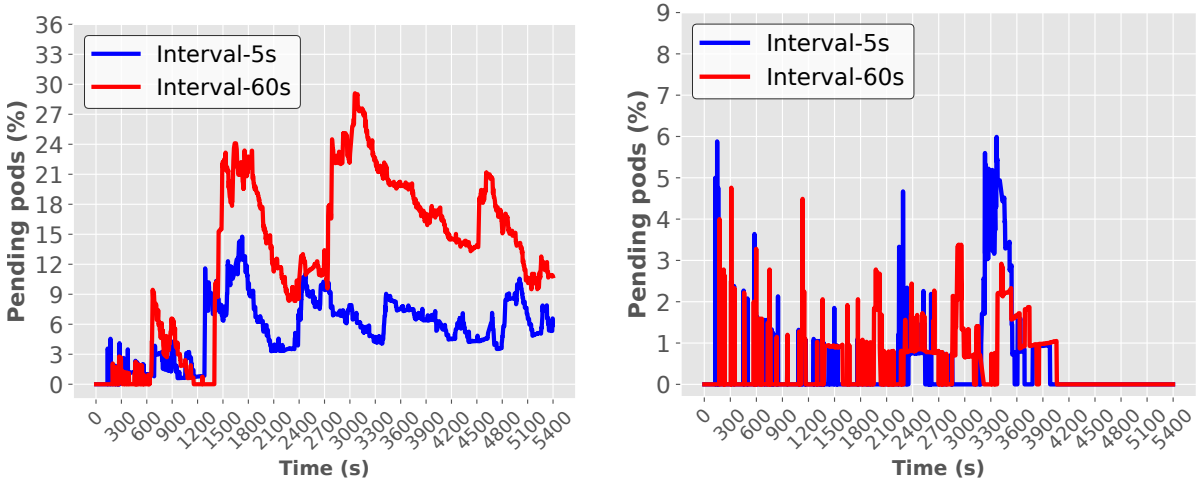
network traffic while achieving a comparable task completion rate to the Prometheus Federation.

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

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 [31]. However, the *fixed* scrape interval in both Prometheus and Prometheus Federation can lead to resource wastage. When the workload 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 [30]. 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 [31] on the management cluster to manage member clusters. mck8s relies on 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 interval for Prometheus Federation may affect the scheduling accuracy when we inject workloads. Two workloads based on Google cluster-usage traces [132] are used. One dataset represents high resource usage for the platform and is identical to the Google cluster-usage traces. In the second dataset, we skip the deployments with even indexes, which therefore generates low resource utilization. We inject each workload for 60 minutes and wait 30 minutes to release the computing resources, resulting in 1,096 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 5.13(a) shows the results of one of these 10 experiment rounds (number 10 in the Table) when we inject high workload, and Table 5.1 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 60 seconds, which indicates that the method schedule can base its placement decisions on high-quality data. However, the cross-cluster network traffic of these two



(a) High workload injection.

(b) Low workload injection.

Figure 5.13 – Percentage of pending Pods with high workload injection (a) and low workload injection (b).

Table 5.1 – 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

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. 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 5.13(b). 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 AdapPF: 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

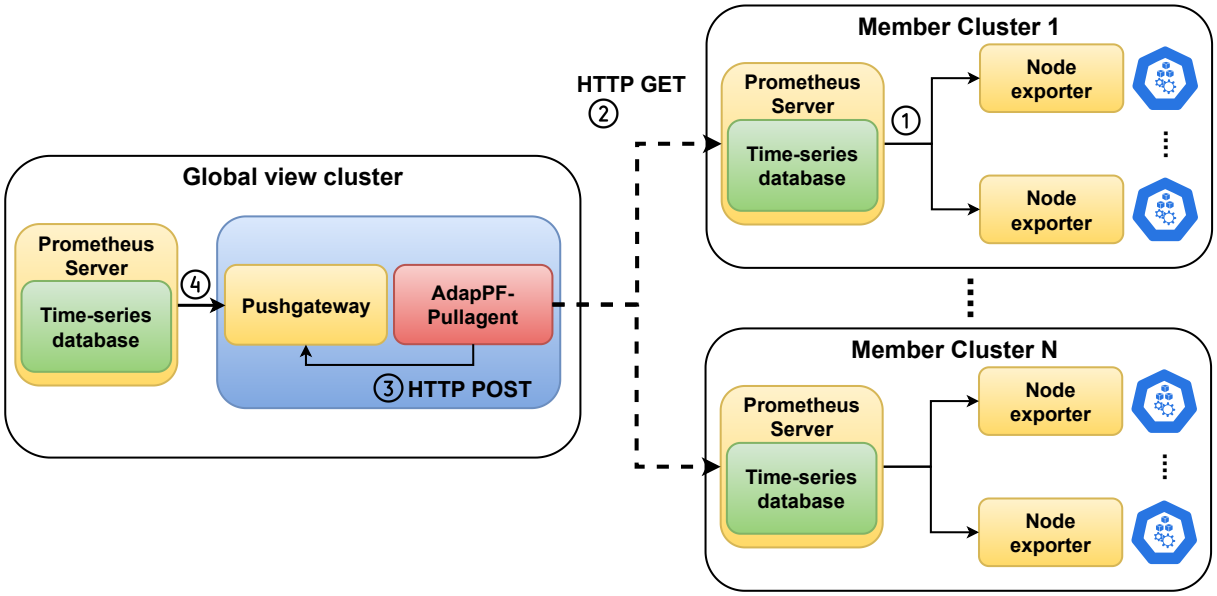


Figure 5.14 – Overview of AdapPF architecture and system workflow.

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.

5.3.1 System Design

The goal of the Adaptive Prometheus Federation (AdapPF) 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 AdapPF and introduces dynamically adjusting scrape interval strategies specifically designed for AdapPF to achieve our objectives.

5.3.1.1 System Model and Architecture

A fog computing platform is a decentralized paradigm with a large number of fog nodes that are typically weak and unstable. To address these characteristics, 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 chapter, 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. Note that AdapPF is not based on the Acala framework because AdapPF aims to provide a solution for users who do not want to aggregate the data in geo-distributed cluster federation environments. Moreover, without metrics aggregation, we can understand the real impact between scrape interval, cross-cluster network traffic, and data accuracy. We leverage the Prometheus server, Node-exporter [122], and Pushgateway [188] to build AdapPF. We present the overview of the architecture in Figure 5.14 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:

(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.

(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.

(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.

(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.

5.3.1.2 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.

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.

Analyze: The main purpose of the Analyze step is to process monitoring data that are collected 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.

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:

$$M = \frac{Tmin_{cluster} - Tmax_{cluster}}{Rmax_{cluster} - Rmin_{cluster}} \tag{5.1}$$

$$intercept = Tmax_{cluster} - M \times Rmin_{cluster} \tag{5.2}$$

$$Interval_{cluster} = M \times CS + intercept \tag{5.3}$$

$Tmin_{cluster}$ 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 interval 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:

$$Tmin_{cluster} \leq Interval_{cluster} \leq Tmax_{cluster} \tag{5.4}$$

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.

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.

5.3.2 Performance Evaluation

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

5.3.2.1 Experimental Setup

We conduct the experiments using the Grid'5000 geo-distributed testbed, which consists of ten server clusters located in various cities [30].

5.3.2.2 Implementation and Experiment Deployment

We implement AdapPF-Pullagent in Python 3.10 and follow the design features discussed in Section 5.3.1. 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. This setting enables automatic restart of the software in case of component errors.

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.

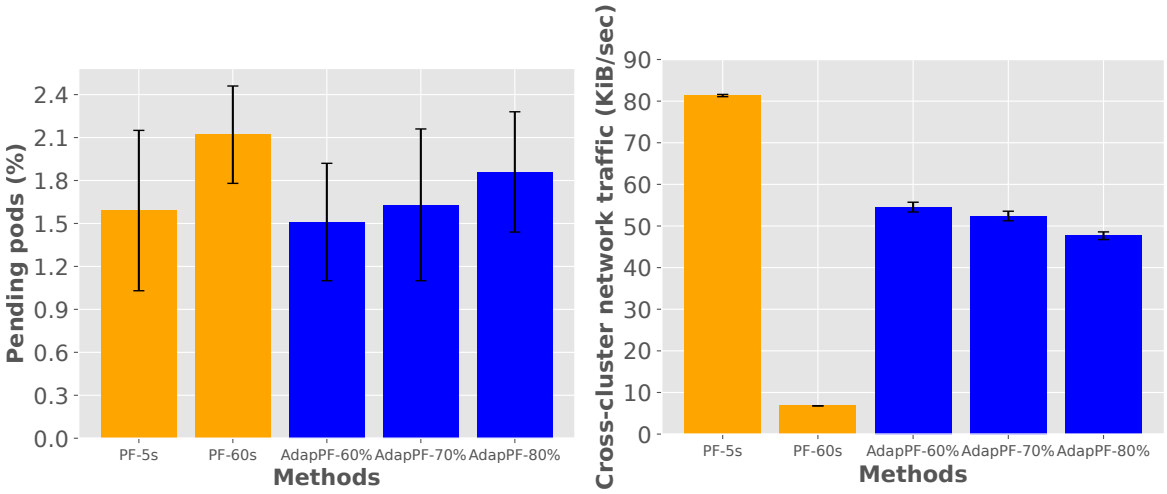
5.3.2.3 Performance Indicators and Test Methods

AdapPF 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 5.3, 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 [191]. We inject workloads (1,096 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.

5.3.2.4 Experiment Results

Figure 5.15 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 $Rmax_{cluster}$.



(a) Percentage of pending Pods

(b) Cross-cluster network traffic

Figure 5.15 – Experiment results of percentage of pending Pods (a) and cross-cluster network traffic (b).

Figure 5.15(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 that 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 5.15(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 5.15(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.

5.4 Conclusion

To support accurate scheduling in a cluster federation while reducing the cross-cluster network traffic generated by Prometheus Federation, this chapter first presents Acala, a monitoring framework for geo-distributed Kubernetes cluster federations. Acala exploits two strategies called metrics aggregation and metrics deduplication for reducing the volume of monitoring data that needs to be reported to the management cluster. Acala performs more efficiently than regular Prometheus Federation because of lower cross-cluster network traffic and shorter scrape duration when we increase the number of worker nodes in a member cluster. We also examine our framework in a federation with large numbers of clusters, proving that the solutions suit the fog environment. Using actual deployments, we show that Acala can reduce the cross-cluster network traffic by up to 95%-97% and scrape duration by up to 55% compared to Prometheus Federation. Moreover, its resource usage remains reasonable and can even save memory resources in the management cluster. Finally, Acala does not significantly impact scheduling efficiency, which shows that report-

ing only aggregated metrics to the management cluster provides an accurate overview for efficient scheduling decisions.

To further address the fixed scrape interval issue, this chapter 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 show that AdapPF achieves comparable accuracy as Prometheus Federation with 5 seconds scrape interval while reducing cross-cluster network traffic by 36%.

Chapter 6

Conclusion and Future Directions

6.1 Conclusion

Cloud computing is a prevailing computing paradigm that has revolutionized the way users access and use computing resources. The cloud architecture consists of many powerful servers and large capacity storage connected by high speed network links in a limited number of data centers. Using virtualization technology, a public cloud data center can share its almost infinite computing resources with many users while maintaining high platform resource utilization. Therefore, cloud users can leverage the public cloud to deploy their applications easily without worrying about the underlying infrastructure. However, the cloud data centers may be far from the users, which increases latency between the users and the cloud applications. This situation may make some types of applications, such as latency-sensitive ones, unsuitable for traditional centralized cloud computing deployment.

The emergence of the fog computing concept aims to address issues such as end-to-end latency, bandwidth constraints, and the need for real-time data processing. The main idea is to deploy computing resources and software at the network edge. Fog computing is regarded as an extension of cloud computing. However, the current fog computing solutions, no matter in the commercial or open-source world, are designed for specific use cases that require the creation of a new dedicated hardware or software infrastructure in the appropriate location to form a private fog platform. Therefore, we argue that future fog platforms should follow the same cloud computing principle that allows users to deploy any application on *large-scale*, *public*, *multi-tenant* geo-distributed fog computing platforms. To build this public fog platform across very large geographical area, engineers would encounter several challenges, such as scalability, resource management, and monitoring. This thesis applies the concepts of cluster federation and assumes each cluster is deployed in different strategic locations to serve the end users. In this context, we then propose two contributions to address the scalability-related challenges.

In the first contribution, this thesis presents the concept of meta-federations, which enables fog clusters to federate their resources with one another in a very flexible way. Each cluster and federation may be owned by a different entity. Peering relationships between clusters may then be established via a legal contract where one company leases some of its hardware resources to another. This design can expand the service coverage of fog providers' geographical span in locations where they do not own resources themselves by using computing resources from other Kubernetes clusters. Using the meta-federation concept potentially allows one to build very large-scale shared public geo-distributed fog platforms at the scale of a country or even a continent, which can then follow a classical cloud-like business model to easily deploy any applications.

To support meta-federations design, this thesis introduces UnBound, a scalable fog meta-federations platform that considers different levels of multi-tenancy to support users and clusters from different organizations in multiple Kubernetes cluster environments. UnBound relies on Kubernetes to orchestrate resources within individual fog clusters and Open Cluster Management (OCM) to federate multiple member clusters under the authority of a management cluster. To isolate the users and federations using the same member cluster, UnBound leverages the Virtual Kubernetes Clusters (vCluster) project to create isolated logical sub-clusters within the member clusters. Each vCluster has its own API server and data store to ensure different federations do not interfere with each other when using the same member cluster.

Extensive experiments with actual large-scale deployments of up to 500 clusters show that UnBound achieves inter-user and inter-federation isolation while maintaining performance comparable to the original Open Cluster Management and acceptable overhead levels.

In the second contribution, we introduce two monitoring frameworks specifically designed for geo-distributed cluster federation environments. The first one is Acala which reports information about entire member clusters rather than the individual servers within them. This elevates the traditional view of federation monitoring from the "node" level to the "cluster" level. Acala exploits two data reduction strategies, metrics aggregation and metrics deduplication, to reduce the number of metrics to be reported to the management cluster. Metrics aggregation aggregates values whose metric name and labels are identical in different servers; and metrics deduplication avoids one to repeatedly report the same metrics in case their value does not change. The evaluations show that Acala can reduce the cross-cluster network traffic by up to 95%-97% and scrape duration by up to 55%

compared to Prometheus Federation. Moreover, its resource usage remains reasonable and can even save memory resources in the management cluster. Finally, Acala does not significantly impact scheduling efficiency, which shows that reporting only aggregate metrics to the management cluster provides an accurate overview for efficient scheduling decisions.

This thesis also discusses Adaptive Prometheus Federation (AdapPF), which can self-adaptive change the scrape interval based on the status of CPU and memory computing resources. If the usage of computing resources is high, AdapPF will automatically increase the frequency of data collection. This allows for the timely acquisition of up-to-date monitoring data, enabling the scheduler or system alarm to make informed decisions or trigger alerts earlier while keeping the cross-cluster network traffic at a lower level. Based on the actual deployments with scheduling experiments, AdapPF achieves comparable scheduling accuracy to Prometheus Federation while reducing cross-cluster network traffic by up to 36%.

Acala and AdapPF are complementary as they address different parts of the Prometheus Federation architecture. In principle, they may be combined to leverage the strengths of both.

6.2 Future Directions

This thesis presents several solutions to address the scalability challenges of geo-distributed fog computing federations. Although we believe these contributions have pushed a step toward building a large-scale, shared, public, geo-distributed fog federation platform, additional research work remains necessary to realize this vision. Therefore, we highlight some future directions in this section and hope they can inspire further research and development in this context.

6.2.1 Automation of Geo-Distributed Fog Computing Federations

Building a fog federation platform requires fog providers to deploy clusters at strategic locations across a country or continent to serve end users efficiently. By leveraging the principles of federation, these fog clusters can operate and manage together, which not only enhances scalability but also enables seamless coordination and resource shar-

ing among clusters. A federation typically consists of a management cluster and many member clusters, which register with the management cluster. To effectively manage a large number of member clusters and workloads in a geo-distributed fog platform, a highly automated and robust framework is essential.

The first direction for automation in large-scale fog federation platforms could focus on infrastructure-level management, such as the registration process between the management cluster and member clusters. To serve a large population in a very broad landscape, the number of member clusters within the federation will need to scale significantly and may reach a very large number. For example, the number of 5G base stations in France is 39,502 with 88.8% of population coverage [22]. To achieve comparable or greater service coverage, the number of member clusters in a fog computing federation would need to be similar to or greater than this number. Moreover, using the meta-federations concept, a fog provider is allowed to expand the service coverage span in additional locations by securing a business deal with the other fog resource providers and including their resources in the federation. Given the complexity of a fog federation platform that includes a large number of member clusters, each of which is potentially managed by a different administrative domain, it is important to investigate automatic mechanisms that make the federations follow the contract to dynamically discover new related clusters, handle registration and de-registration processes, and manage resource quota setting. These automated processes should be able to reduce the potential for human error and the time and effort required for deployment. Moreover, by automating the management of resource quotas, the federation or member cluster's owner could dynamically set resource quotas based on real-time demand, ensuring optimal utilization in each member cluster. To design these automation solutions, a possible direction may be to exploit smart contract technology [192] as a way to formalize and automate to ensure all interactions between the management cluster and member clusters are conducted securely and following predefined agreements.

Another research direction for automation in large-scale fog federation platforms is application-level management, which includes application scheduling and application failover mechanisms. The scheduling methods in current federation frameworks such as UnBound, OCM, mck8s, and Karmada support resource-based placement with labels or cluster names, which provide some automation for deploying the applications across the member clusters. However, addressing specific clusters by their names may become increasingly cumbersome. Moreover, only considering the computing resources of each member cluster would not be enough in this environment since this does not consider fog application char-

acteristics such as low user-to-application latency. Therefore, newer scheduling designs are required that take into account different requirements to support fog applications, such as latency-aware and location-aware placement. An interesting point that also needs to be considered for future solutions is the two-level architecture introduced by the federation, which is composed of the node and cluster levels. For example, the management cluster could first identify a set of suitable member clusters based on the locations or other factors and then find the appropriate servers based on the computing resources or other relevant criteria within those selected clusters.

Classical cloud data centers are composed of powerful servers within stable execution environment conditions. In contrast, geo-distributed fog federation platforms may consist of a very large number of weak and potentially unreliable servers within unstable environments, which introduces a greater chance of facing nodes or cluster failures. Therefore, application failover mechanisms are crucial to maintain service continuity in such platforms. As discussed earlier, the fog federation platform is a two-tier architecture that needs to consider nodes and clusters at the same time when designing failover solutions, which could have different failover levels. Additionally, stateful and stateless applications may also need to be taken into account, which could involve data migration for stateful ones across clusters. Another issue that may be encountered is the trigger conditions for failover. These conditions could take into account application availability by monitoring the status of services, the network connectivity of nodes or clusters, or the health of the control plane or worker nodes.

6.2.2 Security of Geo-Distributed Fog Computing Federations

In general, Cloud Service Providers (CSPs) can build a robust and comprehensive physical security ecosystem because of the centralized nature of cloud data centers. For instance, CSPs could protect their data center infrastructures with surveillance cameras and security guards. In terms of network security, they can defend the network with specifically designed machines, such as network firewalls and Intrusion Prevention Systems (IPSs), by filtering traffic, blocking unauthorized access, and detecting and preventing potential attacks in real-time. On the other hand, a fog federation platform relies on a geo-distributed computing paradigm where fog clusters are deployed in many different locations, which may be vulnerable to physical or remote tampering, potentially turning them into malicious clusters or "spies" within the federation. To mitigate this risk, one potential solution could be to apply the zero-trust security model in the fog

federation platform [193]. This model assumes that no cluster in the federation is trusted, no matter the location or previous behavior. Each interaction between member clusters and management clusters is strictly authenticated to reduce the chances of compromised clusters affecting the federations. However, while the increased security procedures enhance protection, this model may also introduce trade-offs in operational efficiency. This potential trade-off requires a comprehensive evaluation to ensure that the benefits of a higher security level do not reduce the overall system performance.

In a fog meta-federations platform, fog providers can seamlessly access and share their computing resources across different entities. However, servers within each fog provider's cluster may not have the same security conditions. This requires a solution that provides a unified secure computing environment for user applications. One possible strategy is to apply the concept of Trusted Execution Environments (TEEs) [194]. By enabling TEEs in each server, they provide isolated environments that maintain data confidentiality and integrity, thereby offering a standardized level of security across all fog servers in the federation. However, effective TEEs typically rely on hardware support, such as dedicated processors or Trusted Platform Modules (TPMs) [195]. A challenge arises when servers may not be equipped with specialized hardware components. This limitation needs to be considered when designing the solution.

Distributed Denial of Service (DDoS) attack is a significant cybersecurity threat that computing platforms face today. According to a report from Cloudflare, in the year 2023, they have mitigated around 14 million DDoS attacks [196]. Moreover, the number of attacks is still increasing. Therefore, it would be beneficial to investigate a DDOS defense framework specially designed for geo-distributed fog federation platforms. For example, one application in a member cluster may be under DDOS attacks. The security framework should mitigate the DDOS attacks in that cluster while starting to prevent similar attacks for other member clusters in the same federation. Implementing such a framework requires a multi-cluster network policies solution, in which policies are created in one cluster and seamlessly extended to other clusters in the federation. A framework such as Cilium could be part of the defense solution. Cilium is one of the Container Network Interface (CNI) plugins that offer Pod connectivity with support for multi-cluster networking in cloud-native environments [133]. In addition to providing connectivity, Cilium supports multi-cluster network policies, allowing for the creation of global network policies across distributed clusters. However, the effectiveness of these capabilities in a large-scale geo-

distributed fog federation platform requires further evaluation to ensure comprehensive performance.

6.2.3 Sustainability of Geo-Distributed Fog Computing Federations

To deliver high-quality fog services, a fog service provider should deploy fog clusters in rural areas with low-density populations for its own fog federation platform. The computing resources in these clusters may be difficult to fully utilize, which results in some nodes in the cluster remaining idle during non-peak periods. To optimize energy consumption, one could design a self-adaptive mechanism to dynamically and temporarily shut down these idle nodes through the cluster's control plane. This mechanism could follow the Monitor, Analyze, Plan, Execute (MAPE) pattern to monitor node status, analyze node usage patterns in the cluster, plan shutdown timing, and execute the necessary application migration and shutdown actions. It could also combine machine learning or other related technology to further optimize this process by accurately predicting usage patterns for decision-making. Moreover, it could not only shutdown the individual nodes within a cluster but also the entire cluster when necessary. In this case, a remote wake-up protocol for entire clusters by the management cluster is required, which should also be considered in the solutions.

Due to the geo-distributed nature of the fog federation platform, fog clusters are deployed across different locations. The placement of fog clusters is an important factor, as it could significantly influence system performance and environmental sustainability. Fog providers should carefully select the fog cluster locations in order to reduce carbon emissions while keeping the computing resources close to data sources. These location decisions may require multi-criteria analysis to evaluate and compare the environmental impacts of different cluster deployments. To this end, a series of mathematical models is essential to accurately quantify and predict the operational carbon footprint and overall system performance across various deployment scenarios, which may include the considerations of carbon emissions, population density, the availability of infrastructure, and other important factors. By simulating and analyzing potential impacts before real-world implementation, these models may help reduce environmental impacts and assist in identifying and minimizing potential operational, financial, and technological risks.

6.3 Closing Statement

This thesis has proposed the concept of meta-federations to increase the coverage of fog services and improve the utilization of computing resources by sharing the cluster with other fog providers. The UnBound framework addresses the multi-tenancy challenges caused by the meta-federations while maintaining the scalability of the platform. Two monitoring solutions significantly improve the overall monitoring efficiency in cluster federation environments and maintain cost-effectiveness. These contributions pave the way toward the vision of developing large-scale, public, multi-tenant, geo-distributed fog computing platforms and democratizing fog computing technologies.

Bibliography

[1] International Business Machines Corporation, *What Are the Benefits of Cloud Computing?*, https://reurl.cc/WR70R7, cited Mar 2024.

[2] O. Laadan and J. Nieh, « Operating System Virtualization: Practice and Experience », *in Proceedings of the ACM Annual Haifa Experimental Systems Conference*, 2010.

[3] Google LLC, *Cloud Locations*, https://reurl.cc/GjdOLG, cited Apr 2024.

[4] D. Bermbach, F. Pallas, D. G. Pérez, P. Plebani, M. Anderson, R. Kat, and S. Tai, « A Research Perspective on Fog Computing », *in Proceedings of the ICSOC Workshop on IoT Systems Provisioning and Management for Context-Aware Smart Cities*, 2018.

[5] F. Blair, *The Digital Forecast: 40-Plus Cloud Computing Stats and Trends to Know in 2023*, https://reurl.cc/8v30j7, cited Apr 2024.

[6] Z. Wu and H. V. Madhyastha, « Understanding the Latency Benefits of Multi-Cloud Webservice Deployments », *ACM SIGCOMM Computer Communication Review*, vol. 43, *2*, 2013.

[7] C. Avasalcai, I. Murturi, and S. Dustdar, « Edge and Fog: A Survey, Use Cases, and Future Challenges », *in* 2020.

[8] K. Mania, B. D. Adelstein, S. R. Ellis, and M. I. Hill, « Perceptual Sensitivity to Head Tracking Latency in Virtual Environments with Varying Degrees of Scene Complexity », *in Proceedings of the ACM Symposium on Applied Perception in Graphics and Visualization*, 2004.

[9] P. Mark, S. Jason, and T. Christopher, *What's New With the Internet of Things?*, https://reurl.cc/va5Zdy, cited Apr 2024.

[10] L. S. Vailshery, *Number of Internet of Things (IoT) Connected Devices Worldwide from 2019 to 2030*, https://reurl.cc/2YrL69, cited Apr 2024.

[11] L. Hou, S. Zhao, X. Xiong, K. Zheng, P. Chatzimisios, M. S. Hossain, and W. Xiang, « Internet of Things Cloud: Architecture and Implementation », *IEEE Communications Magazine*, vol. 54, *12*, 2016.

[12] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini, I. M. Al Jawarneh, and A. Zanni, « How Fog Computing Can Support Latency/Reliability-Sensitive IoT Applications: An Overview and a Taxonomy of State-of-the-Art Solutions », *in* 2020.

[13] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, « Fog Computing and Its Role in the Internet of Things », *in Proceedings of the ACM SIGCOMM Workshop on Mobile Cloud Computing*, 2012.

[14] Microsoft Corporation, *Azure Stack Edge*, https://reurl.cc/MOo8zk, cited June 2024.

[15] Google LLC, *Google Distributed Cloud Connected*, https://reurl.cc/yLaYYl, cited June 2024.

[16] Oracle Corporation, *Roving Edge Infrastructure*, https://reurl.cc/YEz7Eo, cited July 2024.

[17] Amazon Web Services Inc., *Amazon Web Services Snowball Edge Compute*, https://reurl.cc/oRGDrg, cited July 2024.

[18] D. Battulga, M. Farhadi, M. A. Tamiru, L. Wu, and G. Pierre, « LivingFog: Leveraging Fog Computing and LoRaWAN Technologies for Smart Marina Management (Experience Paper) », *in Proceedings of the Conference on Innovation in Clouds, Internet and Networks*, 2022.

[19] FogGuru project, *The LivingFog Platform*, https://reurl.cc/lQOW0l, cited June 2024.

[20] The KubeEdge Authors, *KubeEdge*, https://reurl.cc/kOR9WK, cited July 2024.

[21] K. Toczé and S. Nadjm-Tehrani, « The Necessary Shift: Toward a Sufficient Edge Computing », *IEEE Pervasive Computing*, vol. 23, *2*, 2024.

[22] The European 5G Observatory Authors, *5G Observatory Biannual Report October 2023*, https://reurl.cc/4r70lY, cited June 2024.

[23] Essentra Components, *A Guide to 5G Small Cells and Macrocells*, https://reurl.cc/AjeNY8, cited June 2024.

[24] Cloud Native Computing Foundation, *Kubernetes Maturity Level*, https://reurl.cc/eL1LrW, cited Mar 2024.

[25] Cloud Native Computing Foundation, *Prometheus Maturity Level*, https://reurl.cc/1vXeE9, cited June 2024.

[26] OpenStreetMap France, *OpenStreetMap France*, https://reurl.cc/XGb0oj, cited June 2024.

[27] The Kubernetes Authors, *Kubernetes*, https://reurl.cc/L48ovL, cited Mar 2024.

[28] The Open Cluster Management Authors, *Open Cluster Management*, https://reurl.cc/qrAYxp, cited Mar 2024.

[29] Loft Labs, Inc., *Virtual Kubernetes Clusters Project*, https://reurl.cc/kOE19q, cited May 2024.

[30] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, « Adding Virtualization Capabilities to the Grid'5000 Testbed », *in Cloud Computing and Services Science*, vol. 367, Springer International Publishing, 2013, pp. 3–20.

[31] M. A. Tamiru, G. Pierre, J. Tordsson, and E. Elmroth, « mck8s: An Orchestration Platform for Geo-Distributed Multi-Cluster Environments », *in Proceedings of the International Conference on Computer Communications and Networks*, 2021.

[32] The Prometheus Authors, *Overview*, https://reurl.cc/Vz1DgA, cited May 2024.

[33] The Prometheus Authors, *Federation*, https://reurl.cc/mM3o2W, cited May 2024.

[34] E. Shein, *The Most Important Cloud Advances of the Decade*, https://reurl.cc/A4zqO3, cited Mar 2024.

[35] J. Surbiryala and C. Rong, « Cloud Computing: History and Overview », *in Proceedings of the IEEE Cloud Summit*, 2019.

[36] A. Regalado, *Who Coined Cloud Computing*, https://reurl.cc/E4amxg, cited Mar 2024.

[37] Amazon Web Services Inc., *Announcing Amazon S3 - Simple Storage Service*, https://reurl.cc/prV8al, cited Mar 2024.

[38] Amazon Web Services Inc., *Announcing Amazon Elastic Compute Cloud (Amazon EC2) - Beta*, https://reurl.cc/Z9z8Zp, cited Mar 2024.

[39] Microsoft Corporation, *Windows Azure General Availability*, https://reurl.cc/E4aWVa, cited Mar 2024.

[40] Google LLC, *App Engine 1.6.0 Out of Preview Release*, https://reurl.cc/WRVyO5, cited Mar 2024.

[41] F. Richter, *Amazon Maintains Cloud Lead as Microsoft Edges Closer*, https://reurl.cc/Xqd44e, cited Mar 2024.

[42] Eurostat, *Cloud Computing - Statistics on the Use By Enterprises*, https://reurl.cc/RWE56Z, cited Mar 2024.

[43] J. E. Smith and R. Nair, « The Architecture of Virtual Machines », *IEEE Computer*, vol. 38, *5*, 2005.

[44] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, « Cloud Container Technologies: A State-of-the-Art Review », *IEEE Transactions on Cloud Computing*, vol. 7, *3*, 2017.

[45] A. Badkar, *7 Astonishing Benefits of Virtualization in Cloud Computing*, https://reurl.cc/D4EqDe, cited Mar 2024.

[46] R. Beri and V. Behal, « Cloud Computing: A Survey on Cloud Computing », *Foundation of Computer Science International Journal of Computer Applications*, vol. 111, *16*, 2015.

[47] GlobalDots, *13 Key Cloud Computing Benefits for Your Business*, https://reurl.cc/77mKY1, cited Mar 2024.

[48] P. Mell and T. Grance, « The NIST Definition of Cloud Computing », 2011.

[49] N. Subramanian and A. Jeyaraj, « Recent Security Challenges in Cloud Computing », *Elsevier Computers and Electrical Engineering*, vol. 71, 2018.

[50] Q. Zhang, L. Cheng, and R. Boutaba, « Cloud Computing: State-of-the-Art and Research Challenges », *Springer Journal of Internet Services and Applications*, vol. 1, 2010.

[51] Amazon Web Services Inc., *Amazon EC2*, https://reurl.cc/WRoyoO, cited Mar 2024.

[52] Google LLC, *Google Cloud Storage*, https://reurl.cc/Z9YVO3, cited Apr 2024.

[53] Microsoft Corporation, *Azure Virtual Network*, https://reurl.cc/YVZqla, cited Apr 2024.

[54] Google LLC, *Google App Engine*, https://reurl.cc/zlko26, cited Mar 2024.

[55] Google LLC, *Google Workspace*, https://reurl.cc/g4oWep, cited Mar 2024.

[56] J. Hong, T. Dreibholz, J. A. Schenkel, and J. A. Hu, « An Overview of Multi-Cloud Computing », *in Proceedings of the AINA Workshop on Multi-Clouds and Mobile Edge Computing*, 2019.

[57] International Energy Agency, *Data Centres and Data Transmission Networks*, https://reurl.cc/qr647R, cited Mar 2024.

[58] N. S. Malik and Bloomberg, *With AI Forcing Data Centers to Consume More Energy, Software That Hunts for Clean Electricity Across the Globe Gains Currency*, https://reurl.cc/WR7M55, cited Mar 2024.

[59] Cloud Security Alliance, *The Treacherous 12: Cloud Computing Top Threats in 2016*, https://reurl.cc/E4evD0, cited Mar 2024.

[60] A. Chaudhary, *Cloud Security Threats to Watch Out for in 2023: Predictions and Mitigation Strategies*, https://reurl.cc/g41olN, cited Mar 2024.

[61] B. Schlinker, I. Cunha, Y.-C. Chiu, S. Sundaresan, and E. Katz-Bassett, « Internet Performance from Facebook's Edge », *in Proceedings of the ACM Internet Measurement Conference*, 2019.

[62] Gigspaces, *Amazon Found Every 100ms of Latency Cost Them 1% in Sales*, https://reurl.cc/E4Nzzk, cited Mar 2024.

[63] G. Linden, *Geeking with Greg: Marissa Mayer at Web 2.0.* https://reurl.cc/M4mdXp, cited Mar 2024.

[64] Statista, *Data Volume of Internet of Things (IoT) Connections Worldwide in 2019 and 2025*, https://reurl.cc/80Az0o, cited Mar 2024.

[65] OpenFog Consortium Architecture Working Group, *OpenFog Reference Architecture for Fog Computing*, https://reurl.cc/j3EQ8Z, cited Mar 2024.

[66] IEEE Standard Association, « IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing », *IEEE Std 1934-2018*, 2018.

[67] M. Chiang and T. Zhang, « Fog and IoT: An Overview of Research Opportunities », *IEEE Internet of Things Journal*, vol. 3, *6*, 2016.

[68] A. Ahmed, H. Arkian, D. Battulga, A. J. Fahs, M. Farhadi, D. Giouroukis, A. Gougeon, F. Gutierrez, G. Pierre, P. Souza Junior, M. A. Tamiru, and L. Wu, « Fog Computing Applications: Taxonomy and Requirements », *arXiv preprint arXiv:1907.11621*, 2019.

[69] A. Ahmed and G. Pierre, « Docker Container Deployment in Fog Computing Infrastructures », *in Proceedings of the IEEE International Conference on Edge Computing*, 2018.

[70] A. van Kempen, T. Crivat, B. Trubert, D. Roy, and G. Pierre, « MEC-ConPaaS: An Experimental Single-Board Based Mobile Edge Cloud », *in Proceedings of the IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, 2017.

[71] C. Wöbker, A. Seitz, H. Mueller, and B. Bruegge, « Fogernetes: Deployment and Management of Fog Computing Applications », *in Proceedings of the IEEE/IFIP Network Operations and Management Symposium*, 2018.

[72] N. Mohamed, J. Al-Jaroodi, I. Jawhar, H. Noura, and S. Mahmoud, « UAVFog: A UAV-Based Fog Computing for Internet of Things », *in Proceedings of the IEEE Smart World Congress*, 2017.

[73] C. Zhu, G. Pastor, Y. Xiao, and A. Ylajaaski, « Vehicular Fog Computing for Video Crowdsourcing: Applications, Feasibility, and Challenges », *IEEE Communications Magazine*, vol. 56, *10*, 2018.

[74] C.-K. Huang and S.-H. Shen, « Enabling Service Cache in Edge Clouds », *ACM Transactions on Internet of Things*, vol. 2, *3*, 2021.

[75] C.-K. Huang, S.-H. Shen, C.-Y. Huang, T.-L. Chin, and C.-A. Shen, « S-cache: Toward an Low Latency Service Caching for Edge Clouds », *in Proceedings of the ACM MobiHoc Workshop on Pervasive Systems in the IoT Era*, 2019.

[76] A. S. Alfakeeh and M. A. Javed, « Intelligent Data-Enabled Task Offloading for Vehicular Fog Computing », *MDPI Applied Sciences*, vol. 13, *24*, 2023.

[77] B. Tang, Z. Chen, G. Hefferman, S. Pei, T. Wei, H. He, and Q. Yang, « Incorporating Intelligence in Fog Computing for Big Data Analysis in Smart Cities », *IEEE Transactions on Industrial Informatics*, vol. 13, *5*, 2017.

[78] N. Chen, Y. Chen, X. Ye, H. Ling, S. Song, and C.-T. Huang, « Smart City Surveillance in Fog Computing », *in* 2017.

[79] Y. Kalyani and R. Collier, « A Systematic Survey on the Role of Cloud, Fog, and Edge Computing Combination in Smart Agriculture », *MDPI Sensors*, vol. 21, *17*, 2021.

[80] H. A. Alharbi and M. Aldossary, « Energy-Efficient Edge-Fog-Cloud Architecture for IoT-Based Smart Agriculture Environment », *IEEE Access*, vol. 9, 2021.

[81] Y. Lin and H. Shen, « CloudFog: Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Service », *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, *2*, 2016.

[82] Fortinet, Inc., *What Is A Data Center?*, https://reurl.cc/Dj14y6, cited Apr 2024.

[83] Fortinet, Inc., *Data Center Security*, https://reurl.cc/VzaN1Z, cited Apr 2024.

[84] Y. Xing and Y. Zhan, « Virtualization and Cloud Computing », *in Proceedings of the Future Wireless Networks and Information Systems*, 2012.

[85] A. Bhardwaj and C. R. Krishna, « Virtualization in Cloud Computing: Moving from Hypervisor to Containerization — A Survey », *Springer Arabian Journal for Science and Engineering*, vol. 46, *9*, 2021.

[86] Docker, Inc., *Registry*, https://reurl.cc/OGlLqX, cited Mar 2024.

[87] A. M. Joy, « Performance Comparison Between Linux Containers and Virtual Machines », *in Proceedings of the International Conference on Advances in Computer Engineering and Applications*, 2015.

[88] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, « A Comparative Study of Containers and Virtual Machines in Big Data Environment », *in Proceedings of the IEEE International Conference on Cloud Computing*, 2018.

[89] C. Pahl, « Containerization and the PaaS Cloud », *IEEE Cloud Computing*, vol. 2, *3*, 2015.

[90] K. Jin and E. L. Miller, « The Effectiveness of Deduplication on Virtual Machine Disk Images », *in Proceedings of the ACM The Israeli Experimental Systems Conference*, 2009.

[91] A. Ahmed, A. Mohan, G. Cooperman, and G. Pierre, « Docker Container Deployment in Distributed Fog Infrastructures with Checkpoint/Restart », *in Proceedings of the IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, 2020.

[92] A. Ahmed and G. Pierre, « Docker-pi: Docker Container Deployment in Fog Computing Infrastructures », *Inderscience International Journal of Cloud Computing*, vol. 9, *1*, 2020.

[93] L. Civolani, G. Pierre, and P. Bellavista, « FogDocker: Start Container Now, Fetch Image Later », *in Proceedings of the IEEE/ACM International Conference on Utility and Cloud Computing*, 2019.

[94] J. Luo, L. Yin, J. Hu, C. Wang, X. Liu, X. Fan, and H. Luo, « Container-Based Fog Computing Architecture and Energy-Balancing Scheduling Algorithm for Energy IoT », *Elsevier Future Generation Computer Systems*, vol. 97, 2019.

[95] P. Bellavista and A. Zanni, « Feasibility of Fog Computing Deployment Based on Docker Containerization Over RaspberryPi », *in Proceedings of the IEEE International Conference on Distributed Computing and Networking*, 2017.

[96] A. Omar, B. Imen, S. M'hammed, B. Bouziane, and B. David, « Deployment of Fog Computing Platform for Cyber Physical Production System Based on Docker Technology », *in Proceedings of the International Conference on Applied Automation and Industrial Diagnostics*, 2019.

[97] E. Yigitoglu, M. Mohamed, L. Liu, and H. Ludwig, « Foggy: A Framework for Continuous Automated IoT Application Deployment in Fog Computing », *in Proceedings of the IEEE International Conference on AI and Mobile Services*, 2017.

[98] Docker, Inc., *Swarm Mode Overview*, https://reurl.cc/XqdbOg, cited Mar 2024.

[99] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, « Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center », *in Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, 2011.

[100] S. Conway, *Survey Shows Kubernetes Leading As Orchestration Platform*, https://reurl.cc/77gA2b, cited Mar 2024.

[101] Datadog, Inc., *9-Insights On Real-World Container Use*, https://reurl.cc/G4W4Rv, cited Mar 2024.

[102] Cloud Native Computing Foundation, *CNCF Survey 2019*, https://reurl.cc/j3LY3p, cited Mar 2024.

[103] Z. Wan, Z. Zhang, R. Yin, and G. Yu, « KFIML: Kubernetes-Based Fog Computing IoT Platform for Online Machine Learning », *IEEE Internet of Things Journal*, vol. 9, *19*, 2022.

[104] W.-S. Zheng and L.-H. Yen, « Auto-Scaling in Kubernetes-Based Fog Computing Platform », *in Proceedings of the International Computer Symposium*, 2019.

[105] N. D. Nguyen, L.-A. Phan, D.-H. Park, S. Kim, and T. Kim, « ElasticFog: Elastic Resource Provisioning in Container-Based Fog Computing », *IEEE Access*, vol. 8, 2020.

[106] The Kubernetes Authors, *Kubernetes Releases and Contributors*, https://reurl.cc/97MydV, cited Apr 2024.

[107] D. Ongaro and J. Ousterhout, « In Search of an Understandable Consensus Algorithm », *in Proceedings of the USENIX Annual Technical Conference*, 2014.

[108] Cloud Native Computing Foundation, *etcd Maturity Level*, https://reurl.cc/rvDGDr, cited August 2024.

[109] The Kubernetes Authors, *What Is a Pod*, https://reurl.cc/G47z6A, cited Apr 2024.

[110] The Kubernetes Authors, *Custom Resources*, https://reurl.cc/WRpnA5, cited Apr 2024.

[111] The Open Cluster Management Authors, *ManifestWork*, https://reurl.cc/YEkyQX, cited June 2024.

[112] The Kubernetes Authors, *Controller Pattern*, https://reurl.cc/Req4VG, cited September 2024.

[113] P. Arcaini, E. Riccobene, and P. Scandurra, « Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation », *in Proceedings of the IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2015.

[114] J. Dobies and J. Wood, *Kubernetes Operators: Automating the Container Orchestration Platform*. O'Reilly Media, 2020.

[115] The Open Cluster Management Authors, *Hub-Spoke Architecture*, https://reurl.cc/0vo00x, cited Apr 2024.

[116] The Kubernetes Authors, *Considerations for Large Clusters*, https://reurl.cc/ezvrYK, cited Apr 2024.

[117] The Kubernetes Authors, *Multicluster Special Interest Group*, https://reurl.cc/Xq4dx0, cited June 2024.

[118] The About API Authors, *About API*, https://reurl.cc/93Gm5v, cited August 2024.

[119] The Work API Authors, *Work API*, https://reurl.cc/QELKdo, cited August 2024.

[120] The KubeFed Authors, *KubeFed: Kubernetes Cluster Federation*, https://reurl.cc/9RGa8x, cited Mar 2024.

[121] The Kubernetes Authors, *Multi-Tenancy*, https://reurl.cc/RWrRyg, cited Mar 2024.

[122] The Prometheus Authors, *Node-exporter*, https://reurl.cc/GjpbAp, cited June 2024.

[123] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, « All One Needs To Know About Fog Computing And Related Edge Computing Paradigms: A Complete Survey », *Elsevier Journal of Systems Architecture*, vol. 98, 2019.

[124] A. J. Fahs and G. Pierre, « Tail-Latency-Aware Fog Application Replica Placement », *in Proceedings of the International Conference on Service-Oriented Computing*, 2020.

[125] P. Souza Junior, D. Miorandi, and G. Pierre, « Stateful Container Migration in Geo-Distributed Environments », *in Proceedings of the IEEE International Conference on Cloud Computing Technology and Science*, 2020.

[126] P. Souza Junior, D. Miorandi, and G. Pierre, « Good Shepherds Care for Their Cattle: Seamless Pod Migration in Geo-Distributed Kubernetes », *in Proceedings of the IEEE International Conference on Fog and Edge Computing*, 2022.

[127] H. Arkian, G. Pierre, J. Tordsson, and E. Elmroth, « Model-Based Stream Processing Auto-Scaling in Geo-Distributed Environments », *in Proceedings of the International Conference on Computer Communications and Networks*, 2021.

[128] J. Huang, C. Xiao, and W. Wu, « Rlsk: A Job Scheduler for Federated Kubernetes Clusters Based on Reinforcement Learning », *in Proceedings of the IEEE International Conference on Cloud Engineering*, 2020.

[129] D. Kim, H. Muhammad, E. Kim, S. Helal, and C. Lee, « TOSCA-Based and Federation-Aware Cloud Orchestration for Kubernetes Container Platform », *MDPI Applied Sciences*, vol. 9, *1*, 2019.

[130] B. Shayesteh, C. Fu, A. Ebrahimzadeh, and R. H. Glitho, « Automated Concept Drift Handling for Fault Prediction in Edge Clouds Using Reinforcement Learning », *IEEE Transactions on Network and Service Management*, vol. 19, *2*, 2022.

[131] F. Faticanti, D. Santoro, S. Cretti, and D. Siracusa, « An Application of Kubernetes Cluster Federation in Fog Computing », *in Proceedings of the Conference on Innovation in Clouds, Internet and Networks*, 2021.

[132] C. Reiss, J. Wilkes, and J. L. Hellerstein, « Google Cluster-Usage Traces: Format + Schema », Google LLC, Tech. Rep., 2011.

[133] The Cilium Authors, *Cilium*, https://reurl.cc/yL5Eva, cited July 2024.

[134] L. Larsson, H. Gustafsson, C. Klein, and E. Elmroth, « Decentralized Kubernetes Federation Control Plane », *in Proceedings of the IEEE/ACM International Conference on Utility and Cloud Computing*, 2020.

[135] The Karmada Authors, *Karmada*, https://reurl.cc/RyW9rz, cited Mar 2024.

[136] M. Iorio, F. Risso, A. Palesandro, L. Camiciotti, and A. Manzalini, « Computing Without Borders: The Way Towards Liquid Computing », *IEEE Transactions on Cloud Computing*, vol. 11, *3*, 2023.

[137] The Virtual Kubelet Authors, *Virtual Kubelet*, https://reurl.cc/mORK1Y, cited Mar 2024.

[138] Loft Labs, Inc., *Kiosk*, https://reurl.cc/GKn2qW, cited Mar 2024.

[139] The Kubernetes Authors, *Resource Quotas*, https://reurl.cc/WRaekL, cited Mar 2024.

[140] Clastix Labs, *Capsule*, https://reurl.cc/OGyOyR, cited Mar 2024.

[141] Kubernetes Multitenancy Working Group, *The Hierarchical Namespace Controller*, https://reurl.cc/QZV7AZ, cited June 2024.

[142] B. C. Şenel, M. Mouchet, J. Cappos, T. Friedman, O. Fourmaux, and R. McGeer, « Multitenant Containers as a Service (CaaS) for Clouds and Edge Clouds », *IEEE Access*, vol. 11, 2023.

[143] Clastix Labs, *Kamaji*, https://reurl.cc/OMLbg7, cited June 2024.

[144] Loft Labs, Inc., *vCluster Becomes First Certified Kubernetes Distribution for Virtual Kubernetes Clusters*, https://reurl.cc/4j1YpK, cited May 2024.

[145] K. Manaouil and A. Lebre, « Kubernetes and the Edge? », Inria Rennes – Bretagne Atlantique, Tech. Rep. RR-9370, 2020.

[146] The Karmada Authors, *Test Report on Karmada's Support for 100 Large-Scale Clusters*, https://reurl.cc/Wv8Emy, cited Mar 2024.

[147] Cloud Native Computing Foundation, *Who We Are*, https://reurl.cc/Dlm6zN, cited August 2024.

[148] Cloud Native Computing Foundation, *Karmada Maturity Level*, https://reurl.cc/9vdxNV, cited July 2024.

[149] Cloud Native Computing Foundation, *Open Cluster Management Maturity Level*, https://reurl.cc/NQEjD9, cited June 2024.

[150] Cloud Native Computing Foundation, *Capsule Maturity Level*, https://reurl.cc/kO4vNd, cited July 2024.

[151] Cloud Native Computing Foundation, *Certified Kubernetes Software Conformance*, https://reurl.cc/ezOZxm, cited July 2024.

[152] J. Povedano-Molina, J. M. Lopez-Vega, J. M. Lopez-Soler, A. Corradi, and L. Foschini, « DARGOS: A Highly Adaptable and Scalable Monitoring Architecture for Multi-Tenant Clouds », *Elsevier Future Generation Computer Systems*, vol. 29, *8*, 2013.

[153] Zabbix LLC, *Zabbix*, https://reurl.cc/GjNl43, cited June 2024.

[154] S. A. De Chaves, R. B. Uriarte, and C. B. Westphall, « Toward an Architecture for Monitoring Private Clouds », *IEEE Communications Magazine*, vol. 49, *12*, 2011.

[155] D. Trihinas, G. Pallis, and M. D. Dikaiakos, « JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud », *in Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2014.

[156] Nagios, *Nagios*, https://reurl.cc/MOjkRW, cited June 2024.

[157] B. Costa, J. Bachiega Jr, L. R. Carvalho, M. Rosa, and A. Araujo, « Monitoring Fog Computing: A Review, Taxonomy and Open Challenges », *Elsevier Computer Networks*, vol. 215, 2022.

[158] S. K. Battula, S. Garg, J. Montgomery, and B. Kang, « An Efficient Resource Monitoring Service for Fog Computing Environments », *IEEE Transactions on Services Computing*, vol. 13, *4*, 2019.

[159] M. Abderrahim, M. Ouzzif, K. Guillouard, J. Francois, and A. Lebre, « A Holistic Monitoring Service for Fog/Edge Infrastructures: A Foresight Study », *in Proceedings of the IEEE International Conference on Future Internet of Things and Cloud*, 2017.

[160] H. G. Abreha, C. J. Bernardos, A. D. L. Oliva, L. Cominardi, and A. Azcorra, « Monitoring in Fog Computing: State-of-the-Art and Research Challenges », *Inderscience International Journal of Ad Hoc and Ubiquitous Computing*, vol. 36, *2*, 2021.

[161] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, and V. Stankovski, « Monitoring Self-Adaptive Applications within Edge Computing Frameworks: A State-of-the-Art Review », *Elsevier Journal of Systems and Software*, vol. 136, 2018.

[162] M. Großmann and C. Klug, « Monitoring Container Services at the Network Edge », *in Proceedings of the International Teletraffic Congress*, 2017.

[163] Tildeslash Ltd., *Monit*, https://reurl.cc/oRyx7Q, cited June 2024.

[164] Á. Brandón, M. S. Pérez, J. Montes, and A. Sanchez, « FMonE: A Flexible Monitoring Solution at the Edge », *Hindawi Wireless Communications and Mobile Computing*, vol. 2018, *1*, 2018.

[165] Mesosphere, Inc., *Marathon*, https://reurl.cc/3X97vX, cited June 2024.

[166] S. Forti, M. Gaglianese, and A. Brogi, « Lightweight Self-Organising Distributed Monitoring of Fog Infrastructures », *Elsevier Future Generation Computer Systems*, vol. 114, 2021.

[167] M. Gaglianese, S. Forti, F. Paganelli, and A. Brogi, « Assessing and Enhancing a Cloud-IoT Monitoring Service Over Federated Testbeds », *Elsevier Future Generation Computer Systems*, vol. 147, 2023.

[168] V. Colombo, A. Tundo, M. Ciavotta, and L. Mariani, « Towards Self-Adaptive Peer-to-Peer Monitoring for Fog Environments », *in Proceedings of the International Conference on Software Engineering for Adaptive and Self-Managing Systems*, 2022.

[169] S. Ilager, J. Fahringer, S. C. de Lima Dias, and I. Brandić, « DEMon: Decentralized Monitoring for Highly Volatile Edge Environments », *in Proceedings of the IEEE/ACM International Conference on Utility and Cloud Computing*, 2022.

[170] S. Ilager, J. Fahringer, A. Tundo, and I. Brandić, « A Decentralized and Self-Adaptive Approach for Monitoring Volatile Edge Environments », *arXiv preprint arXiv:2405.07806*, 2024.

[171] A. Aznavouridis, K. Tsakos, and E. G. M. Petrakis, « Micro-Service Placement Policies for Cost Optimization in Kubernetes », *in Proceedings of the International Conference on Advanced Information Networking and Applications*, 2022.

[172] G. Carcassi, J. Breen, L. Bryant, R. W. Gardner, S. McKee, and C. Weaver, « SLATE: Monitoring Distributed Kubernetes Clusters », *in Proceedings of the ACM Practice and Experience in Advanced Research Computing*, 2020.

[173] Y.-W. Chan, H. Fathoni, H.-Y. Yen, and C.-T. Yang, « Implementation of a Cluster-Based Heterogeneous Edge Computing System for Resource Monitoring and Performance Evaluation », *IEEE Access*, vol. 10, 2022.

[174] T. Hu and Y. Wang, « A Kubernetes Autoscaler Based on Pod Replicas Prediction », *in Proceedings of the Asia-Pacific Conference on Communications Technology and Computer Science*, 2021.

[175] T. Lin, S. Marinova, and A. Leon-Garcia, « Towards an End-to-End Network Slicing Framework in Multi-Region Infrastructures », *in Proceedings of the IEEE International Conference on Network Softwarization*, 2020.

[176] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, « Machine Learning-Based Scaling Management for Kubernetes Edge Clusters », *IEEE Transactions on Network and Service Management*, vol. 18, *1*, 2021.

[177] J. Cho and Y. Kim, « A Design of Serverless Computing Service for Edge Clouds », *in Proceedings of the International Conference on Information and Communication Technology Convergence*, 2021.

[178] T. Dockendorf, T. Baer, and D. Johnson, « Early Experiences with Tight Integration of Kubernetes in an HPC Environment », *in Proceedings of the ACM Practice and Experience in Advanced Research Computing*, 2022.

[179] C.-K. Huang and G. Pierre, « UnBound: Multi-Tenancy Management in Scalable Fog Meta-Federations », *in Proceedings of the 17th IEEE/ACM International Conference on Utility and Cloud Computing*, Dec. 2024.

[180] Kubernetes SIGs, *Kubernetes Metrics Server*, https://reurl.cc/RrmAGG, cited June 2024.

[181] S. Hemminger, « Network Emulation with NetEm », *in Proceedings of the Australia's National Linux Conference*, 2005.

[182] The Kubernetes Authors, *Resource Management for Pods and Containers*, https://reurl.cc/70d3k1, cited June 2024.

[183] Google LLC, *Online Boutique*, https://reurl.cc/z670ga, cited June 2024.

[184] The Kubernetes Authors, *Kubernetes in Docker*, https://reurl.cc/70d3R1, cited June 2024.

[185] C.-K. Huang and G. Pierre, « Acala: Aggregate Monitoring for Geo-Distributed Cluster Federations », *in Proceedings of the 38th ACM/SIGAPP Symposium On Applied Computing*, Tallinn, Estonia, Mar. 2023.

[186] C.-K. Huang and G. Pierre, « AdapPF: Self-Adaptive Scrape Interval for Monitoring in Geo-Distributed Cluster Federations », *in Proceedings of the 28th IEEE Symposium on Computers and Communications*, Tunis, Tunisia, Jul. 2023.

[187] C.-K. Huang and G. Pierre, « Aggregate Monitoring for Geo-Distributed Kubernetes Cluster Federations », *IEEE Transactions on Cloud Computing*, vol. 12, *4*, 2024.

[188] The Prometheus Authors, *Pushgateway*, https://reurl.cc/mMyvDj, cited June 2024.

[189] The Kubernetes Authors, *Workloads*, https://reurl.cc/9Gajkn, cited June 2024.

[190] The stress-ng Authors, *stress-ng*, https://reurl.cc/p3vx6l, cited June 2024.

[191] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, « Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis », *in Proceedings of the ACM Symposium on Cloud Computing*, 2012.

[192] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, « An Overview on Smart Contracts: Challenges, Advances and Platforms », *Elsevier Future Generation Computer Systems*, vol. 105, 2020.

[193] Y. He, D. Huang, L. Chen, Y. Ni, and X. Ma, « A Survey on Zero Trust Architecture: Challenges and Future Trends », *Wiley Wireless Communications and Mobile Computing*, vol. 2022, *1*, 2022.

[194] M. Sabt, M. Achemlal, and A. Bouabdallah, « Trusted Execution Environment: What It Is, and What It Is Not », *in Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2015.

[195] S. L. Kinney, *Trusted Platform Module Basics: Using TPM in Embedded Systems*. Elsevier, 2006.

[196] Cloudflare, Inc., *DDoS Threat Report for 2024 Q2*, https://reurl.cc/XRlog0, cited August 2024.

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Scalabilité des Fédérations de Fog Computing Publiques Géo-distribuées

Mot clés : Scalabilité, Fog Computing, Fédérations, Multi-Tenancy, Surveillance

Résumé : Construire une plateforme de fog computing publique, géo-distribuée, multi-tenant et à grande échelle, où n'importe quelle application peut être déployée, nécessite un grand nombre de ressources de calcul placées à différents endroits stratégiques couvrant un pays entier ou même un continent. L'un des défis pour réaliser cette plateforme publique de fog est la scalabilité. À cette fin, cette thèse se concentre sur la résolution de certains défis liés à l'évolutivité et propose une série de solutions. Tout d'abord, nous présentons le concept de méta-fédérations, où de nombreux fournisseurs de ressources locaux indépendants peuvent louer leurs ressources à plusieurs fournisseurs de fog afin de résoudre les problèmes de couverture de service et d'utilisation des ressources. Nous proposons UnBound, un système scalable de meta-federations qui aborde spécifiquement les défis difficiles du multi-tenancy introduits par les méta-fédérations. Ensuite, nous proposons deux systèmes de surveillance conçus pour les environnements de fédération de clusters géo-distribués, Acala et AdapPF, qui visent à réduire le trafic réseau inter-cluster de la surveillance tout en maintenant la précision des données de surveillance.

Title: Scalability of Public Geo-Distributed Fog Computing Federations

Keywords: Scalability, Fog Computing, Federations, Multi-Tenancy, Monitoring

Abstract: Building a large-scale, multi-tenant, public, geo-distributed fog computing platform where any application can be deployed requires a large number of computing resources placed at different strategic locations spanning an entire country or even a continent. One of the challenges to realizing this public fog platform is scalability. To this end, this thesis focuses on addressing some scalability challenges and proposes a series of solutions. First, we present the meta-federations concept, where many independent local resource providers may lease their resources to multiple fog providers to solve the service coverage and resource utilization issues. We propose UnBound, a scalable meta-federations framework that specifically addresses the difficult multi-tenancy challenges introduced by meta-federations. Second, we propose two monitoring frameworks designed for geo-distributed cluster federation environments, Acala and AdapPF, which aim to reduce the cross-cluster network traffic of monitoring while maintaining the accuracy of the monitoring data.