MkRP: Multiple k Registry Placement for Fast Container Deployment on Edge Computing

CHUNGGEON SONG , HEONCHANG YU * , Joon-Min Gil *

Posted Date: 1 February 2024

doi: 10.20944/preprints202402.0064.v1

Keywords: container deployment; registry placement; edge computing

Article

MkRP: Multiple k Registry Placement for Fast Container Deployment on Edge Computing

Chunggeon Song [1], Heonchang Yu [1,*] and Joon-Min Gil [2,*]

[1] Department of Computer Science and Engineering, Korea University, Seoul 02841, Korea; security0730@korea.ac.kr (C.G.); yuhc@korea.ac.kr (H.Y.)
[2] Department of Computer Engineering in School of Software, College of Engineering, Jeju National University, 63243, Korea; jmgil@jejunu.ac.kr
* Correspondence: yuhc@korea.ac.kr (H.Y.); jmgil@jejunu.ac.kr (J.G.)

Abstract: Edge computing reduces the response time of real-time services with handling dynamic traffics reliably. Edge servers with limited resources utilize container technology that provides a lightweight execution environment. When deploying containers in edge servers, a container image is required and is downloaded from a remote registry. Therefore, these operations are dependent on network overhead between the container deployment system and the remote registry. Through motivation experiments, we show that container pooling time increases in proportion to physical distance and has characteristics that vary flexibly depending on runtime time. In this study, we define a system model for high-speed registry deployment in this edge system and propose a clustering technique into k groups based on the network overhead and affinity of the regionally distributed edge servers that make up the edge system. Also, considering idle resources, we propose a technique to deploy a registry after electing a leader for each cluster. A simulation experiment is conducted to verify the performance of the proposed technique and shows that performance improvement can be achieved regardless of the number of edge servers and the k value.

Keywords: container deployment; registry placement; edge computing

1. Introduction

Edge computing is a computing paradigm in which edge servers located near users perform some of the tasks that central cloud servers used to perform in order to conserve network resources and reduce service response times. Pervasive IoT devices are increasing the amount of valuable data, and the rise of real-time services that leverage sensor data in applications such as augmented reality, automotive, healthcare, and disaster recovery is driving the importance of edge. In particular, healthcare services process health data continuously collected from patients' devices through edge computing, enabling rapid response to emergency situations [3, 4].

Multiple edge servers at the end of a network for edge computing have relatively limited resources compared to the central cloud, and require technologies that can reliably handle the fluid traffic generated by device mobility. To meet these requirements for edge servers, container virtualization, which provides a lightweight execution environment, and cloud-native technologies to run scalable services are being leveraged and are considered de-facto standards.

Each edge server has a service scope of a certain size, running some tasks of real-time services for clients within its scope, or performing preprocessing on data generated by sensor devices located nearby. In addition, to provide multi-tenancy on edge servers, container operating platforms will utilize very different types of container images in short period of time. In particular, when executing a task based on FaaS on an edge server, the work unit size becomes smaller and the type and number of containers required to execute the function increase.

If the container image required for container deployment is not available locally, image pulling is performed to download the container image from a remote registry. However, due to the nature of edge computing, edge servers are geographically distributed over long distances and are connected to a wide area network (WAN). Accordingly, network overhead has a significant impact on container

 doi:10.20944/preprints202402.0064.v1

deployment speed. As a result, research on container placement techniques considering network overhead emerged [5-13].

Knob et al. [5] proposed a technique for modeling the network topology between edge servers in the form of a weighted graph, clustering it based on the fluid communities algorithm, and distributing a registry for each cluster. Temp et al. [6] proposed a strategy to deploy a container registry considering user mobility. Roges et al. [7] proposed a policy policy that dynamically provisions the registry according to the storage of the server and the needs of the application. Because these traditional techniques because traditional techniques only perform one policy or one clustering technique, there is a low probability of finding container images in registries deployed nearby. Additionally, it has the disadvantage of inconsistent container deployment time because it does not take into account dynamic changes in network traffic.

In this study, we propose a technique for dividing edge servers into k groups and deploying registry servers to improve the deployment speed of containers operating in a system performing edge computing. The proposed technique performs two-stage registry placement based on RTT-based network overhead and affinity-based similarity of the container images used. It also covers techniques for distributing container registries based on idle resource information and container usage frequency within each cluster. The contributions of this study are as follows.

- We present a two-stage clustering technique that classifies edge servers based on network overhead and affinity and a registry deployment technique.
- We present a two-stage clustering technique that classifies edge servers based on network overhead and affinity and a registry deployment technique.
- Through simulation, we confirmed that the proposed algorithm shows stable performance under various number of edge servers and number of clusters k.

The rest of this paper is as follows. Section 2 explains the background of the research, and Section 3 explains the proposed k registry placement technique for fast container deployment. Then, Section 4 describes simulation-based experiments to demonstrate the performance of MkRP. Section 5 describes related work on container deployment and registry placement, and finally, Section 6 summarizes the contributions of the paper and explains future research.

2. Background

We redefined and modeled the registry deployment problem to improve container deployment speed in a distributed system environment performing edge computing. In this chapter, we describe the edge system architecture that is commonly cited in defining the registry server deployment problem and describe a motivation experiment to check the network overhead that affects the container deployment speed.

Multiple regionally separated edge servers have wired and wireless networks beyond WAN, and data communication between two edge servers goes through many router hops. In communication through a WAN network, the data transmission and reception path can be determined depending on the routing algorithm, and since all network lines have limited bandwidth, the data movement speed can be affected by traffic. Figure 1 conceptually represents various types of network connection relationships between two edge servers.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 1 February 2024 doi:10.20944/preprints202402.0064.v1

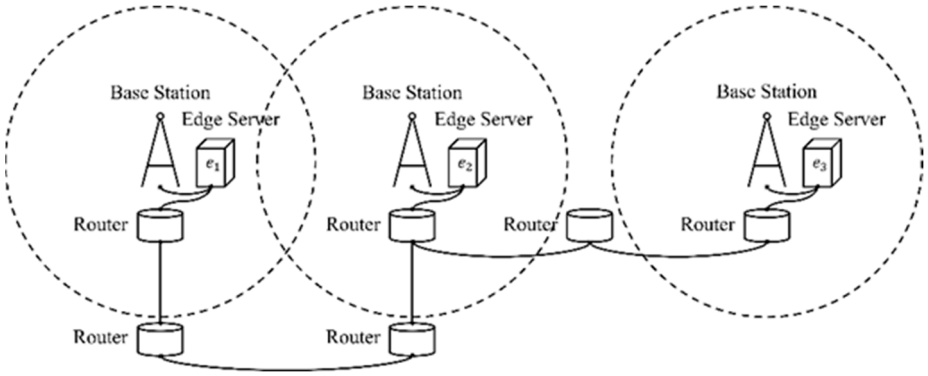


Figure 1. Types of network connections between two edge servers.

Both wireless and wired communication is possible between the edge servers, e1 and e2. The wireless connection is connected directly to the overlapping communication range of the Base Station, and the wired connection is connected through four router hops. Additionally, wireless communication is not possible between edge servers e2 and e3, and only wired communication via 3 router hops is possible. This communication method between edge servers makes it difficult to model network overhead, and makes network traffic fluid over time due to various factors.

Based on this system model, we performed a motivation experiment to specifically observe the network overhead arising from data communication between edge servers. We used the Azure public cloud service to deploy multiple registry servers in different countries with different physical distances. And we measured the speed of image pooling on a computer(client) within Korea University The registry server is located in the same country as the client (kr), a nearby country (jp), and another continent(us). All registry servers utilized the same template of VM. Docker 24.0.4 version was used as the container runtime, and the ubuntu 20.04 was used as the container image. Image pooling was performed 5 times and the average was calculated. Figure 2 shows image pooling time according to network distance.

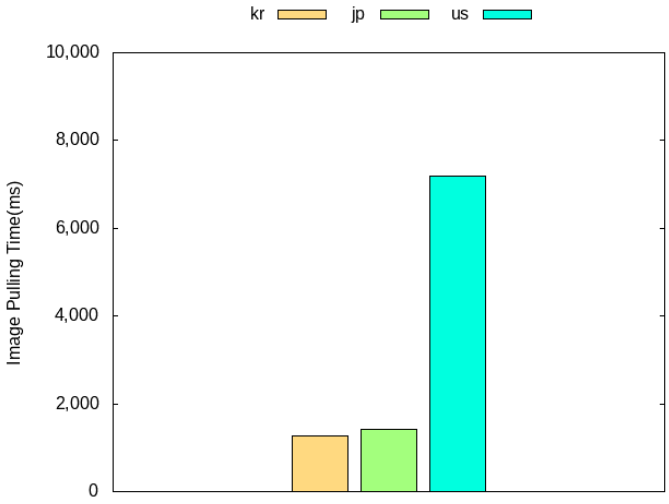


Figure 2. Image pulling time depending on network distance.

Experimental result shows the image pulling time is increased in proportion to the physical distance between the client and the registry server. Through these experimental results, it was confirmed that when pulling container images from a registry server located in a short distance, the container deployment time can be reduced. Then, to check the liquidity of network overhead generated during container pooling, an experiment was conducted to measure the Round Trip Time

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 1 February 2024 doi:10.20944/preprints202402.0064.v1

(RTT) between the client and the registry server at 1 second intervals. Figure 3 shows the results of measuring the RTT between the client and three registry servers for 12 hours in the same environment as the first motivation experiment.

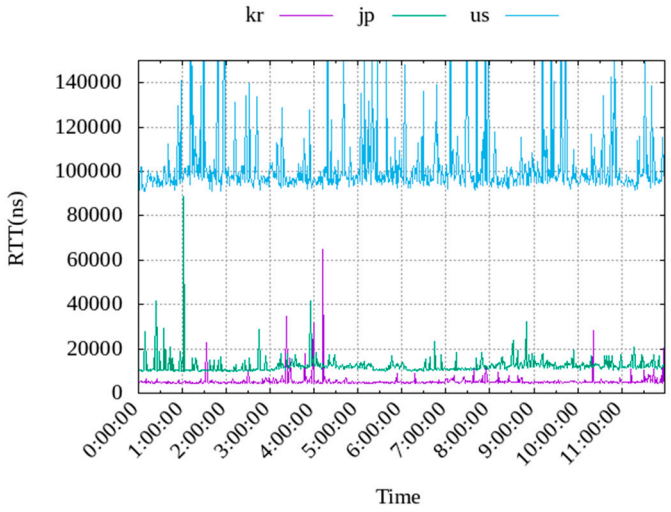


Figure 3. RTT changes over time.

Experimental result shows some overall difference in RTT proportional to the physical distance and great liquidity of RTT over time. In particular, between 4 o'clock and 5 o'clock, the RTT for the kr registry server was higher than that for the jp registry server regardless of the physical distance due to the impact on network traffic. These results confirmed that network overhead changes over time, and that periodic network overhead monitoring is necessary to improve container deployment.

3. Proposed Solution

3.1. System Model

This section describes various system models for quantifying the speed of image pooling involved in container deployment on an edge server with limited resources. In particular, it defines the components and metrics that affect the image pulling time of multiple edge servers connected by a network over a WAN. Table 1 describes various symbols used for defining a system model.

Table 1. Notations.

Symbol	Description
E	$\{e_i \mid e_i$ is i^{th} edge server, $1 \le i \le N\}$
R	$\{r_k \mid r_k$ is k^{th} registry, $1 \le j \le K\}$
$S_{i,j}^{RTT}$	Some of RTTs occurred between e_i and e_j Collected during Time Interval M
S^{RTT}	Set of $S_{i,j}^{RTT}$, $1 \le i,j \le N$
S_i^{image}	Set of Sample Data for Image Request in e_i Collected during Time Interval M
S^{image}	Set of S_i^{image}, $1 \le i \le N$
$I(e_i)$	The ratio of idle resources for e_i
$O(e_i, e_j)$	Network Overhead between e_i and e_j
$A(e_i, e_j)$	Affinity between e_i and e_j

$$O(e_i, e_j) = \frac{\sum_{i=1}^{|S_{i,j}^{RTT}|} a_i}{2|S_{i,j}^{RTT}|}, a_i \in S_{i,j}^{RTT} \quad (1)$$

 doi:10.20944/preprints202402.0064.v1

Equation 1 represents the formula for calculating the network overhead between edge servers e_i and e_j, $O(e_i, e_j)$. This requires $N!$ calculations per M cycles. The average of the RTT data collected through message communication between e_i and e_j is calculated and divided by 2 to obtain the one-way delay time.

$$I(e_i) = 1 - (cpuUsage_i * \alpha + diskUsage_i * \beta) \quad (2)$$

Equation 2 represents the formula for calculating the idle resource ratio of e_i. An environment in which all edge servers are homogeneous is assumed, and $cpuUsage_i$ and $diskUsage_i$ used in $I(e_i)$ have values between 0 and 1. α and β are weight values that control which resource type is given more weight when calculating the ratio of idle resources for each edge server.

$$A(e_i, e_j) = \frac{|S_i^{image} \cap S_j^{image}|}{|S_i^{image} \cup S_j^{image}|} \quad (3)$$

Equation 3 represents the formula for calculating the affinity between edge servers e_i and e_j. This value is calculated as the ratio of the number of commonly pooled image sets to the total number of image sets requested from edge servers e_i and e_j.

3.2. MkRP

The multiple k registry deployment algorithm, which is the core technique of the proposed MkRP, collects idle resource information and affinity of edge servers every M times and operates based on the two metrics, and is expressed in Algorithm 1. Clustering is performed only when k is greater than N/2. This algorithm utilizes G^O, which defines $O(e_i, e_j)$ as a complete graph, and G^A, which defines $A(e_i, e_j)$ as a complete graph, between all edge servers every M cycles. *ClusterGraph*() divides the complete graph received as an argument into k subgroups based on the index number of the edge server. *DeploymentRegistry*() deployments the registry service to the edge server. Container images stored in the registry are selected based on recently used container image log data from the Leader Edge server.

Algorithm 1 multiple k registry deployment algorithm

Input: G^O, G^A, k
01: *leader* ← *null*
02: /* first clustering */
03: $\{G_i\}_{i \in I}$ ← *ClusterGraph*(G^O, k)
04: **for** G_i **in** $\{G_i\}_{i \in I}$
05: *leader* ← *ElectLeader*(G_i, *'networkOverhead'*)
06: **if** *leader* ≠ *null*
07: *DeploymentRegistry*(*leader*)
08: **end if**
09: **end for**
10: /* second clustering */
11: $\{G_i\}_{i \in I}$ ← *ClusterGraph*(G^A, k)
12: **for** G_i **in** $\{G_i\}_{i \in I}$
13: *leader* ← *ElectLeader*(G_i, *'affinity'*)
14: **if** *leader* ≠ *null*
15: *DeploymentRegistry*(*leader*)
16: **end if**
17: **end for**

In Algorithm 1, line 3 expresses the operation of clustering the complete graph for network overhead into k subgroups. Loop in lines 4-9 expresses the feature of deployment the registry server by electing a leader among the edge servers in the subgroup based on network overhead. Line 11 expresses the second clustering operation, dividing the complete graph of affinity into k subgroups.

 doi:10.20944/preprints202402.0064.v1

The loop in lines 12-17 performs the function of electing a leader among edge servers and distributing a registry server based on the complete affinity graph. The *ElectLeader*() function in lines 5 and 13 is expressed in detail in Algorithm 2.

Algorithm 2 leader election algorithm

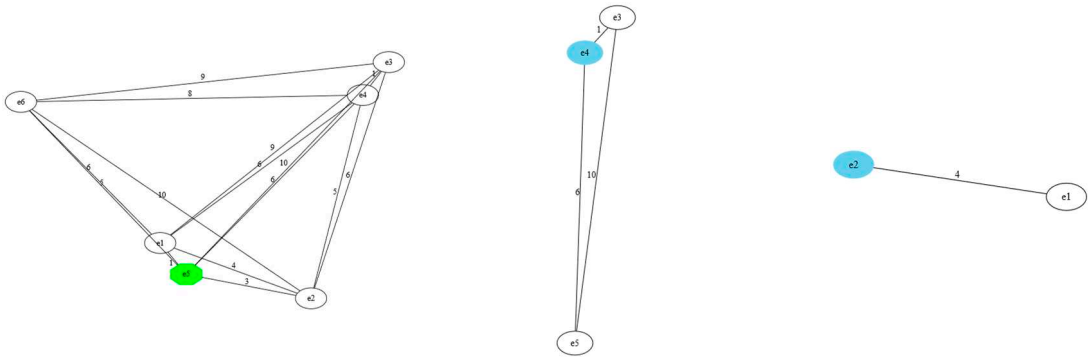
Input: G, *graphType*
Output: e
01: *leader* ← *null*
02: *minSum* ← 100000
03: *maxSum* ← 0
04: *sum* ← 0
05: **for** v_i **in** G
06: *sum* ← 0
07: **for** *Path* **in** G.*Paths*
08: **if** *Path is connected to* v_i
09: *sum* ← *sum* + *Path.weight*
10: **end if**
11: **if** $I(e_i) \geq \lambda$
12: **if** *graphType* = *'networkOverhead'*
13: **if** *minSum* ≥ *sum*
14: *leader* ← v_i
15: *minSum* ← *sum*
17: **end if**
18: **else if** *graphType* = *'affinity'*
19: **if** *maxSum* ≤ *sum*
20: *leader* ← v_i
21: *maxSum* ← *sum*
22: **end if**
23: **else**
24 return *null*
25: **end if**
26: **else**
27: return *null*
28: **end if**
29: **end for**
30: **end for**

In Algorithm 2, lines 5-30 perform an operation to find the leader by traversing the graph nodes. Lines 7-29 perform the function of electing a leader by using the sum of the weights between a specific node and the remaining nodes. If the input *graphType* value is '*networkOverhead*' according to lines 12-17, the node with the minimum weight sum is elected as the leader. If the *graphType* value is '*affinity*', the node with the maximum weight sum is elected as the leader according to lines 18-22. In line 11, the ratio of idle resources is greater than the threshold λ, it is elected as the leader.

Every registry stores a certain number of container images, and container images are identified by ID. In the simulation, the option for the number of edge servers is set manually, and which container to deploy is determined using a weight-based random value. Container image pulling time is calculated based on the network overhead between the container deployment system and the registry that holds the requesting container image. Figure 4 shows an example of a complete graph representing the network overhead generated in a certain resource management cycle when the number of edge servers is set to 5 and the simulation is performed. (a) in Figure 4(a) represents the entire graph, and the light green hexagonal nodes represent the closest routers to all edge nodes. Figure 4(b) and Figure 4(c) represent graphs clustered into two subgroups. In this figure, the blue

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 1 February 2024 doi:10.20944/preprints202402.0064.v1

node is the leader of the subgraph and performs the role of a registry. Among the nodes in each subgroup, the node with the minimum sum of O(e_i,e_j) between the remaining modes is selected as the leader.



(a) (b) (c)

Figure 4. Complete graph representing network overhead between edge servers.

4. Evaluation

4.1. Simulator

We developed a simulator with the Go language to verify the performance of MkRP, the proposed registry placement technique. Each edge server has a cache of a certain size, and when pulling images, container images stored in the cache are used preferentially. In addition, one or more registry server information can be registered on the edge server, and when deploying containers, container images are sequentially searched according to the registration order.

Every registry stores a certain number of container images, and container images are identified by ID. In the simulation, the option for the number of edge servers is set manually, and which container to deploy is determined using a weight-based random value. Container image pulling time is calculated based on the network overhead between the container deployment system and the registry that holds the requesting container image. Figure 4 shows an example of a complete graph representing the network overhead generated in a certain resource management cycle when the number of edge servers is set to 5 and the simulation is performed. (a) in Figure 4(a) represents the entire graph, and the light green hexagonal nodes represent the closest routers to all edge nodes. Figure 4(b) and Figure 4(c) represent graphs clustered into two subgroups. In this figure, the blue node is the leader of the subgraph and performs the role of a registry. Among the nodes in each subgroup, the node with the minimum sum of $O(e_i, e_j)$ between the remaining modes is selected as the leader.

Figure 5 shows a complete graph expressing the affinity created in a certain resource management cycle in the same environment. (a) in Figure 5(a) represents the entire graph, and Figure 5(b) and Figure 5(c) represent graphs clustered into two subgraphs. In this figure, the red node is the leader of the subgraph and performs the role of a registry. Among the nodes in each subgroup, the node with the maximum sum of $A(e_i, e_j)$ between the remaining modes is selected as the leader.

 doi:10.20944/preprints202402.0064.v1

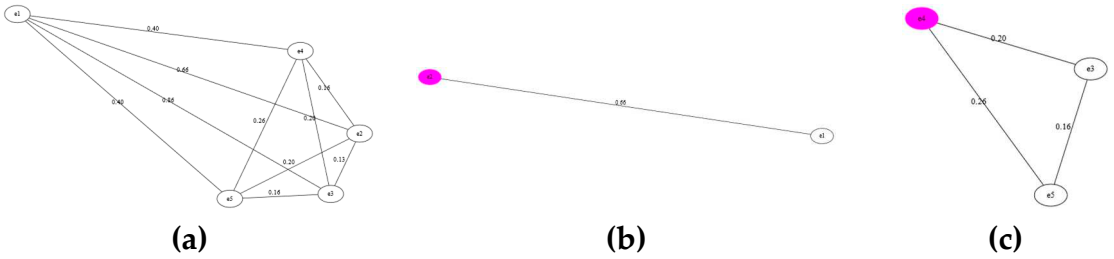


Figure 5. Complete graph representing affinity between edge servers.

4.2. Simulaton scenario

In the simulation experiment to verify the performance of MkRP, the number of registry servers was set to 5, and each registry server stores 100 container images of 500MB capacity. And each edge server has a cache size of 5GB. If the cache capacity is exceeded, the edge server replaces the newly downloaded container image with a less frequently used container. Container images requested from each edge server were requested from a specific registry server three times more frequently than the remaining registry servers. All edge servers request a container image once per second.

Container images are searched in the following order: local cache, G^O and G^A based two registries deployed on the edge server that plays the leader role, and five remote registries. There are three baselines: a case where MkRP is not used (non), a case where only MkRP's Network Overhead-based k registry placement technique is performed (mkrp no), and both MkRP's Network Overhead-based k registry placement technique and Affinity-based k registry placement technique it is performed (mkrp no+af). In the simulation experiment, container images were requested 10,000 times from 5 edge servers and resource management was performed every 1,000 seconds, and the results are shown in Figure 6.

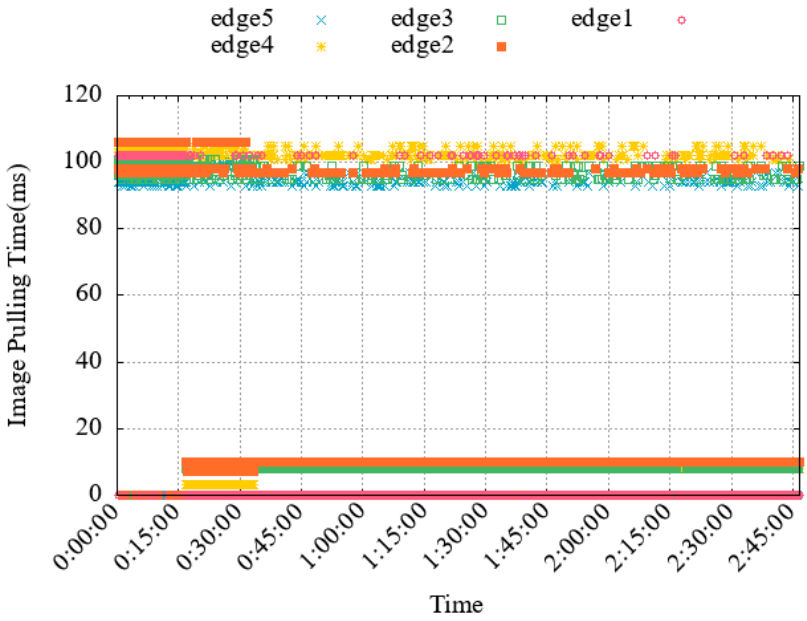


Figure 6. Image pooling simulation

As the resource management technique is applied after 16 minutes, the result shows that the number of references to the container image in the remote registry gradually decreases. Depending on the registry server preference of each edge server, you can see the results of frequently requesting a specific container image. By utilizing container images through a registry built on a nearby edge server, the image pulling time was drastically reduced. Next, we describe different numbers of edge servers and k values and describe an experiment to check the average image pooling time.

 doi:10.20944/preprints202402.0064.v1

4.3. Image pooling performance according to the numbers of edge servers

We conducted an experiment to check the container image pooling performance of MkRP according to changes in the number of edge servers. The image pooling time was measured by setting the k value to 3 and changing the number of edge servers to 10, 20, and 30. In the experiment, resource management was performed at 30 minute intervals for 12 hours, and the results are shown in figure 7.

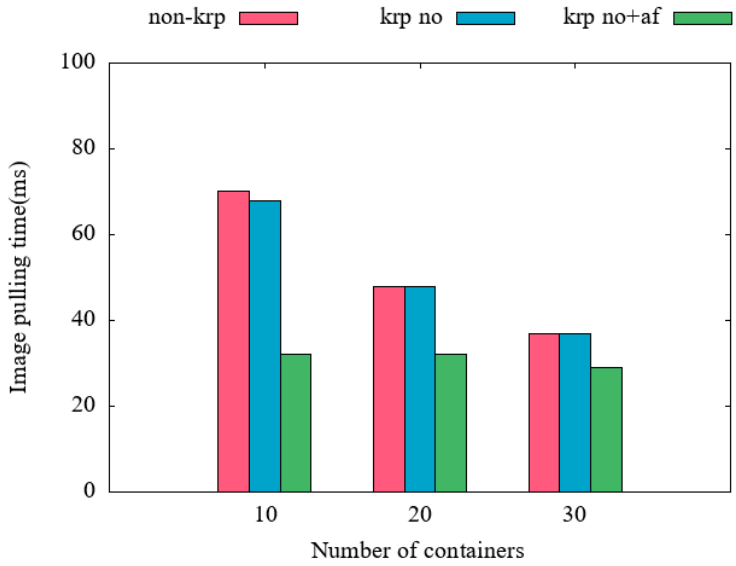


Figure 7. Image pooling time according to the numbers of edge servers.

In experimental result 'mkrp no+af' showed 40% and 39% higher performance than non-mkrp and mkrp no for all edge server numbers. In cases non-mkrp and mkrp no, the average image pooling time decreased in proportion to the number of containers, and the reason for this phenomenon was that the hit rate increased as the frequency of utilizing the local cache of the edge server increased. The case mkrp no+af showed consistent performance regardless of the number of containers.

4.4. Image pooling performance according to various k values

We conducted an experiment to check the container image pooling performance of MkRP according to changes in k, the number of subgroups. In the experiment, the number of edge servers was set to 30 and the image pooling time was measured while changing the k value to 5, 10, and 15. MkRP was operated at 30-minute intervals for 12 hours, and the results are shown in figure 8.

 doi:10.20944/preprints202402.0064.v1

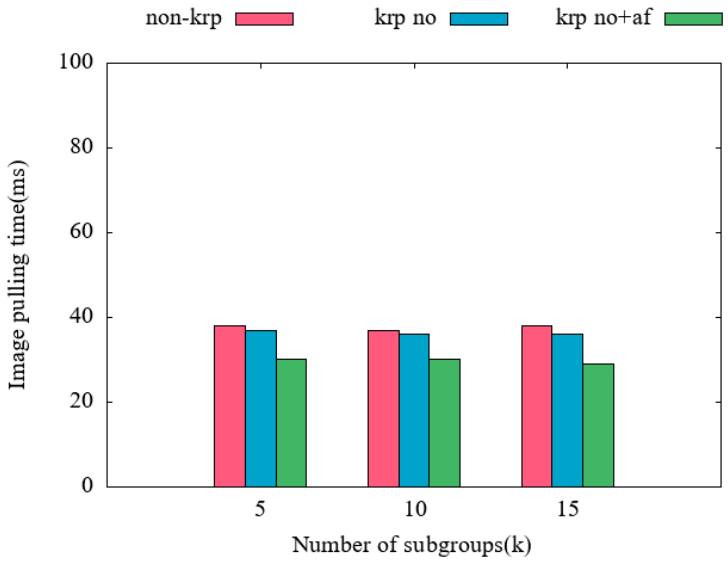


Figure 8. Image polling time for different k values.

The experimental results showed that the edge server had 21% and 18% higher performance for mkrp no and mkrp no+af, respectively, than non-mkrp for all k, and that performance slightly improved in proportion to the k value. However, performance does not increase when the k value exceeds a certain value. As the value of k increases, the number of edge groups increases and the number of registry servers deployed increases, and the rate of referencing remote registries decreases, which is the cause of performance improvement. However, if k increases above a certain number, registry operation overhead increases more than the level of performance improvement, so it is important to set an optimized k.

5. Related Work

5.1. Research on container deployment

Existing research on container deployment is evolving with more than one purpose. Lin et al. [14] proposed a technique to deploy containers for the purpose of energy consumption while satisfying the SLA. Hu et al. [15] proposed a technique to deploy containers by considering load balancing and dependencies between containers while ensuring multi-resource requirements. In order to perform tasks sensitive to response speed on edge servers, it is important to quickly deploy a lightweight execution environment [16]. Accordingly, various studies have emerged to improve container deployment speed. Harter et al. [17] proposed a technique to improve container deployment speed by utilizing a centralized storage server shared between the container deployment system and the container registry.

5.2. Research on container registry placement

Various existing studies have investigated efficient techniques for dividing edge servers into several subgroups and distributing one container registry to each subgroup. Knob et al. [5] modeled the static network topology as a complete graph and developed a clustering technique. Temp et al. [6] proposed a technique for deploying a container registry based on user mobility. However, in situations where network traffic is elastic, these studies show that it is difficult for the complete graph to store accurate network overhead information, and thus the performance improvement rate of the average container deployment time is reduced. Roges et al. [7] proposed a technique for dynamically deploying the registry according to the server's storage size and application requirements. This utilizes dynamic data, but has limitations in selecting a registry leader by utilizing metrics for one type of resource and traffic. The MkRP is differentiated from existing studies in that it utilizes a

combination of network overhead and affinity and increases the probability of finding a container image in a registry distributed nearby.

6. Conclusion

The demand for using containers on edge computing will increase, and the types of images for deploying containers will also become more diverse. In particular, when providing multi-tenancy in cloud services, multiple tenant users frequently deploy various types of services, and it is impossible to provide stable services with a single container registry. For this reason, the importance of additional registry deployment within edge servers will increase.

In this paper, we proposed a technique for dividing edge servers into k groups and deploying registry servers to improve the deployment speed of containers operating in a system performing edge computing. A simulation experiment was conducted to verify the performance of the proposed technique and showed that dramatic performance improvement was observed regardless of the number of edge servers and the k value. In the future, we plan to develop a technique to hierarchically cluster multiple edge servers, elect a leader, and deploy a registry.

Acknowledgments: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT)(No. NRF-2022R1A2C1092934).

References

1. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. Proceedings of the first edition of the MCC workshop on Mobile cloud computing(MCC) ,Helsinki, Finland, 17 August 2012; pp. 13–16.
2. Cao, K.; Liu, Y.; Meng, G.; Sun, Qimeng. An overview on edge computing research. IEEE access 2020, 8, 85714 - 85728.
3. Oueida, S.; Kotb, Y.; Aloqaily, M.; Jararweh, Y.; Baker, Thar. An edge computing based smart healthcare framework for resource management. Sensors 2018, 18, 4307.
4. Ray, P.P.; Dash, D.; De, D. Edge computing for Internet of Things: A survey, e-healthcare case study and future direction. Journal of Network and Computer Applications 2019, 140, 1-22.
5. Knob, L.A.D.; Faticanti, F.; Ferreto, T.; Siracusa, D. Community-based placement of registries to speed up application deployment on edge computing. International Conference on Cloud Engineering (IC2E), Virtual, 4-8 October 2021; pp. 147-153.
6. Temp, D.C.; Souza, P.S.S.D.; Lorenzon, A.F.; Luizelli, M.C.; Rossi, F.D. Mobility-aware registry migration for containerized applications on edge computing infrastructures. Journal of Network and Computer Applications 2023, 217, 103676.
7. Roges, L.; Ferreto, T. Dynamic Provisioning of Container Registries in Edge Computing Infrastructures. Proceedings of the 24th brazilian symposium on high performance computing systems(WSCAD), Porto Alegre/RS, Brazil, 17-20 October 2023; pp. 85-96.
8. Nathan, S.; Ghosh, R.; Mukherjee, T.; Narayanan, K. Comicon: A co-operative management system for docker container images. International Conference on Cloud Engineering (IC2E), Vancouver, Canada, 4-7 April 2017; pp. 116-126.
9. Faticanti, F.; Pellegrini, F.D; Siracusa, D.; Santoro, D.; Cretti, S. Cutting throughput with the edge: App-aware placement in fog computing. International Conference on Edge Computing and Scalable Cloud (EdgeCom), Paris, France, 21-23 June 2019; pp. 196-203.
10. Kangjin, W.; Yong, Y.; Hanmei, L.; Lin, Ma. FID: A faster image distribution system for docker platform. International Workshops on Foundations and Applications of Self* Systems (FAS* W), Tucson, AZ, USA, 18-22 September 2017; pp. 191-198.
11. Littley, M.; Anwar, A.; Fayyaz, H.; Fayyaz, Z.; Tarasov, V.; Rupprecht, L.; Skourtis, D.; Mohamed, M.; Ludwig, H.; Cheng, Y.; Butt, A.R. Bolt: Towards a scalable docker registry via hyperconvergence. International Conference on Cloud Computing (CLOUD), Milan, Italy, 8-13 July 2019; pp. 358-366.
12. Ahmed, A.; Pierre, G. Docker container deployment in fog computing infrastructures. International Conference on Edge Computing (EDGE), San Francisco, CA, USA, 2-7 July 2018; pp. 1-8.
13. Rossi, F.; Cardellini, V.; Presti, F.L.; Nardelli, M. Geo-distributed efficient deployment of containers with Kubernetes. Computer Communications 2020, 159, 161-174.
14. Lin, C.; Chen, J.; Liu, P.; Wu, J. Energy-efficient core allocation and deployment for container-based virtualization. International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 11-13 December 2018, pp 93-101.

 doi:10.20944/preprints202402.0064.v1

15. Hu, Y.; Laat, C.D.; Zhao, Z. Multi-objective container deployment on heterogeneous clusters. International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Larnaca, Cyprus, 14-17 May 2019, pp 592-599.
16. Darrous, J.; Lambert, T.; Ibrahim, S. On the importance of container image placement for service provisioning in the edge. International Conference on Computer Communication and Networks (ICCCN), Valencia, Spain, 29 July 2019 - 01 August 2019, pp 1-9.
17. Harter, T.; Salmon, B.; Liu, R.; Arpaci-Dusseau, A.C.; Arpaci-Dusseau, R.H. Slacker: Fast distribution with lazy docker containers. USENIX Conference on File and Storage Technologies (FAST 16), Santa Clara, CA, 22-25 February 2016, pp 181-195.