IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 8 December 2022, accepted 27 December 2022, date of publication 29 December 2022, date of current version 11 January 2023.

Digital Object Identifier 10.1109/ACCESS.2022.3233159

RESEARCH ARTICLE

HACM: High Availability Control Method in Container-Based Microservice Applications Over Multiple Clusters

BOOPATHI RAMASAMY, YEONJOO NA, WEONSU KIM, KYOUNGBEOM CHEA, AND JUN KIM
Samsung Electronics, Suwon, Gyeonggi 16677, South Korea

Corresponding author: Boopathi Ramasamy (r.boopathi@samsung.com)

This work was supported in part by Samsung Electronics, Suwon, South Korea.

ABSTRACT Emerging cloud-native technologies, such as container runtime and container orchestrator, offer unprecedented agility in developing and running applications, especially when combined with microservice-style architecture. Several commercial Samsung Network products such as Samsung Element Management System (S-EMS), 5G Radio Access Network (RAN) & Core network elements are being redesigned to fit the microservice paradigm. The cloud environment allows enterprises to scale their applications on-demand with minimum cost; however, it is often difficult to use containers without sacrificing the many benefits offered by container technology. S-EMS manages 5G RAN & Core network elements (NEs) deployed nationwide, and systematically stores a huge volume of stateful data per second. Containers are characterized to have an ephemeral state, hence 'stateful-ness' aspect of S-EMS makes management more complex. The existing system in a container-based application does not support geo-redundancy where services/data are stateful/state dependent. In this paper, different challenges around geo-redundancy between different independent Kubernetes set up with active and standby modes between the sites where state-dependent data is stored in each site are described. To overcome these challenges, we propose the High Availability Control Method (HACM) - which enables the Kubernetes cluster to be active and standby, where state-dependent data and context-based operations are intrinsically supported by the underlying S-EMS container application. Our approach has been designed to maintain the geo-redundancy philosophy of cloud-native by associating the status of each site using high availability (HA), switching over services based on the health of applications, deciding state when there are conflicts in site state, and the option to auto fallback based on user preference and services are transferred between sites without user intervention with optimized storage that ensures consistency, persistence, reliability, and availability. Through evaluation, we show that HACM with S-EMS can facilitate geo-redundancy HA, while not posing a significant burden on the Cloud.

INDEX TERMS 5G networks, microservices, containers, cloud, communication networks, geo-redundancy, and high availability.

I. INTRODUCTION

Container-based microservice architecture has gained substantial attention among several 5G Telco vendors and operators. Many challenges of traditional monolithic architecture applications are tackled by the microservices paradigm [1]. However, to leverage the benefits of the microservices style, one needs to use technologies aligned with the characteristics of microservices [1]. The container runtime and container-orchestrator have become a popular deployment formats for microservice applications. In Kubernetes or container-based environments, applications are deployed as the smallest deployable computing units called pods, an abstract way to expose an application running on a set of the pod as services. The container-based environment is considered an independent cluster where data is independent and stateless.

The associate editor coordinating the review of this manuscript and approving it for publication was Olutayo O. Oyerinde.

However, Element management System (EMS) should manage Network Elements Fault, Configuration, Accounting, Performance and Security (FCAPS) management data into Data Base(DB) and storage so the nature of EMS is stateful, it is a lot different, unlike stateless container application, in such context [3], there is no effective solution provided by the container-based application to keep effective active-active as operation complexity such as Network elements should be distributed among sites, it is error-prone to the operator, data conflict as two sites are running and data should be updated on two sites, but there can be a delay in synchronization which does not guarantee services will work correctly. Thus, maintaining the active-standby mode is preferred by most top-tier network operators. Today, organizations face several challenges that threaten business continuity. While disasters are inevitable, unplanned downtimes disrupt normal business processes, services, operations, and customer dissatisfaction. Delivering good performance and excellent user quality of experience (QoE) are also crucial. All of these reasons are why geo-redundancy between data centers is mission-critical for many businesses.

Several commercial Samsung 5G network products including Samsung Element Management System (S-EMS), radio access Central Unit (CU), radio access Distributed Unit (DU), and 5G Core Network Functions are being redesigned to fit the microservice paradigm as containers. Specifically, S-EMS is characterized to manage FCAPS data on 5G Network Elements (5GNEs) such as Radio Access Network (RAN) and Core elements deployed nationwide. The S-EMS systematically handles and stores a very large amount of stateful data per second. S-EMS handling such a scale of stateful data, and aligning it to fit with the microservice paradigm with geo-redundancy has called-for new research issues and architectural design changes. Aimed at operator manual work of moving network elements where two container-based independent clusters are running with active-standby mode with state-dependent data and service. In existing Kubernetes-based microservice cluster does not support high availability(HA) between sites or clusters and does not provide active or standby modes. It does not have a method to make which site is active and standby and if there are any conflicts in state and option to user to choose preference to make a site active or standby. The active and standby sites do not have the option to synchronize state-dependent data directly so backup is done with an external backup location and data is restored to the standby site when a site is coming up which leads to having extra storage location and time to start services are delayed when state-dependent data size is very large.

In this paper, we introduce HACM — High Availability control method in container-based microservice applications over multiple clusters which enables high availability for state-dependent data and services intrinsically supported by the underlying application running between independent clusters in different geographical locations. Our approach has been designed to high availability pod (HA pod) running at each setup of a cluster which monitors the health of S-EMS application pods and services, handshake with other cluster HA pod, and provides switch over functionality when there is an issue or fault or disaster in site, provides the option to user to make preference to configure site when a site is recovered after the crash to decide whether the running site needs to be in active mode or standby mode. The details of HACM are explained in sections III.

II. RELATED WORK

Microservice architecture has significant benefits in terms of flexibility and scalability, compared with the traditional monolithic architecture. Microservice has been drawing attention in the literature [10]. The HA over geographical with context-based data has been studied for some time by the research community, spanning a wide spectrum such as Content Distribution Networks (CDNs), Information/Content-Centric Networking (ICNs/CCNs), and many other areas. However, our work with HAMC is one of the first attempts to introduce such an HA control method to microservice architecture principles in a cloud-native domain. Looking at other related works, in [11] author investigates microservice coordination among multiple clouds to enable seamless and real-time responses to service requests from mobile users. The objective of this work is to devise the optimal microservice coordination scheme which can reduce the overall service delay with low costs. However, no focus has been given to stateful information retrieval and processing using these microservices. In [12], authors proposed a predictive auto-scaling orchestration system for cloud-native telecom microservices. A predictive mechanism with workload forecasting was proposed to enable auto-scaling and thereby improve the performance of the orchestration systems. However, in this paper, the investigation was performed only for stateless replicas. The stateful replication management is one of the key hurdles when aligning with microservice architectures. In [13], the authors reported on state-dependent data retrieval that can be improved by receiving user interventions such as user feedback, user preferences, and machine context-based data, but this loses its efficiency with millions of users, and context profiles start being part of the system.

In [3], the framework was introduced for contextual information retrieval which enables context-based operations to be intrinsically supported by the underlying application for data placement and retrieval functions in single cluster-based applications. All of the above-mentioned approaches utilize either user behavior or user preferences or both to construct a contextual profiles and some forms of user intervention are required. In addition, none of these approaches discusses how to effectively perform context-based stateful storage and retrieval on a large-scale distributed system. With the existing frameworks, it is not possible to expand the system components independently, without downtime. System components include stateful storage, stateless processing elements, and support for various types of input (for example, 5G network element types). All existing research does not address how to

handle geo-redundancy where data are state-dependent and underlying applications are running at the container-based microservice platform in a separate cluster; there are many research and solutions for stateless applications; however, for stateful nature applications, they are many limitations. In the remainder of this paper, we discuss the HACM framework which enables HA for state-dependent data and services to be intrinsically supported by an underlying application running over between two independent clusters in different geographical locations.

III. HACM FRAMEWORK

Container-based cluster environment is not supported by default as active and standby between sites where data is state-dependent and services are stateful. Active site or cluster will have its network, and network elements to be managed where data from network elements are state-dependent such as alarm, configuration, security data, and other state-dependent data. A standby site or cluster is a backup set up with the same resource configuration as an active site or cluster which is capable of handling services when there is a problem in the active site.

A. HACM ARCHITECTURE

The S-EMS with HACM architecture will be able to seamlessly handle the site by providing an active-standby mode, as shown in Fig.1. A cluster is an independent Kubernetes setup where applications, pods/services are running to manage network elements. The network elements can be 4G, 5G RAN, or core network elements. Here, the communication between network elements and S-EMS applications is represented as Pod A1, A2 . . . An, DB pod and HA pod. Persistent Volume(PV) is a shared storage at the cluster level, where all applications or pods can store data. Master Node is consisting of one more control and data plane to support Kubernetes functionalities at the cluster level. Here, clusters can play the role of active or standby. Active cluster means all services-related network elements functions are running in that cluster. If the cluster is in standby mode means the cluster is waiting for service to start, however, the application or pod will be running as part of or in a standby state and will not provide any services to the end-user when it is in standby mode.

Therefore, when a disaster occurs in a running or active cluster, applications or services should be moved to a standby cluster where data or services are state-dependent. The HACM concept presented in this paper will be covered as an HA pod in a diagram where each cluster is running in a different geographical location. The main concept, functionality, and logic of an HA pod are described in detail in the following sections.

B. HANDSHAKE

The HA pod in each cluster handshake with other HA pods in other clusters. The handshake confirms the health state of the cluster and the HA pod can decide when to switch over to be executed based on the current state of the cluster. The

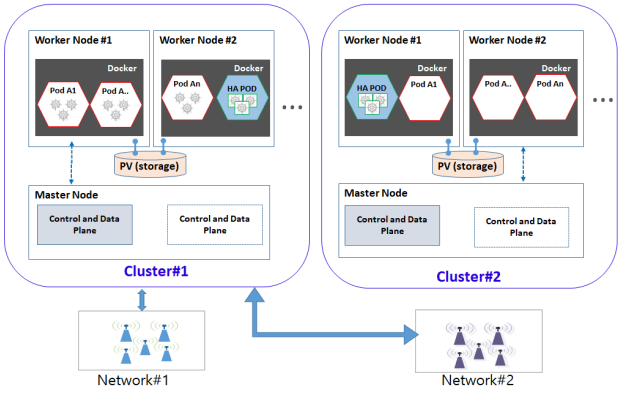

FIGURE 1. HACM architecture.

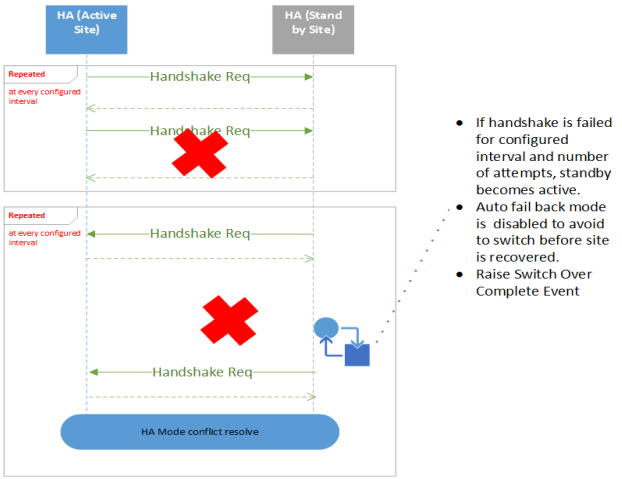

FIGURE 2. Handshake between HA pod.

handshake between HA pods is shown in Fig.2. Handshake between HA pod occurrence using the REST API interface. A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of the REST architectural style and allows interaction with RESTful web services. REST represents representational state transfer. Here, the REST API interface is used to communicate securely over the network Hypertext Transfer Protocol Secure (HTTPS) after authentication and authorization with system-defined accounts. Here, a handshake is performed between the active site HA pod and the standby site HA pod. Both pods independently handshakes. When a handshake between an active and standby HA pod is successful, it is considered normal, and it is not required to take any action. In case the active site HA pod detects handshake failure with standby HA, there is no action taken as the site is in an active state, and the problem is with the standby site.

Here, failure means the destination site is not reachable due to network issues between HA pod clusters or site or application or dependent services are having errors or unable to provide service due to various reasons. As shown in Fig.2,

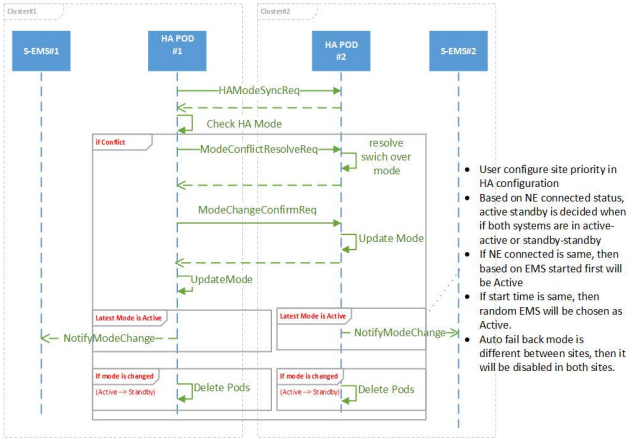

FIGURE 3. HA mode conflict resolve between clusters.

when there is no handshake issue with the active site from the standby site, no action is taken. When the standby site HA pod handshake fails with the active site HA pod, the HA pod needs to change from the standby site to the active site. The switchover is explained in detail in Section III.E. Before switching is determined, the system attempts for the configured number of the interval using configured time. If all retry attempts are failed, then considered a handshake is a failure, otherwise, a handshake is considered a success, and the system will be continuing the functionalities. When a handshake is failed, an event is generated to inform the user that the handshake functionalities are affected and used for auditing purposes.

C. HA MODE CONFLICT RESOLVE

When S-EMS comes to each site, there is a chance that both sides can be in active-active or standby-standby mode. Here, the S-EMS provides a method to handle HA mode conflicts when the system comes up and decides which site needs to be active or on standby based on criteria. Therefore, the HA pod verifies the HA mode when the HA pod is coming up at a respective site. Communication between HA pods running in different clusters occurred through the REST API interface. For example, HA pod 1 in cluster #1 is coming up and HA pod 1 will synchronize HA mode with other HA pod running in cluster #2.

When the modes of both sites are the same, they are called conflicts. In this, the system should have the intelligence to decide which site is active or which site is on standby. Here, HAMC provides a method to solve HA mode conflicts in detail, and how the communication occurs between HA pods is shown in Fig.3. Here all communication between HA pod will happen through the REST API interface. HA pod 1 communicates with HA pod 2 in the other clusters through the REST API.

HA pod 1 requests an HA mode synchronization request(HAModeSyncReq) to HA pod 2 and HA pod 2 returns the current cluster mode. There can be 4 possible cases as shown in Table 1.

TABLE 1. Possible HA mode status between cluster.

Cases	Cluster #1 Status	Cluster #2 Status	Remarks
1	ACTIVE	ACTIVE	Abnormal, HA mode Conflicts
2	STANDBY	STANDBY	Abnormal, HA mode Conflicts
3	ACTIVE	STANDBY	Normal
4	STANDBY	ACTIVE	Normal

Here, Cases 1 and 2 are the problems, and Cases 3 and 4 are normal. Cases 1 and 2 are required to resolve conflicts. HA mode resolve request(ModeConflictsResolveReq) is sent to the HA2 pod by the HA1 pod. The HA2 pod will resolve HA mode as shown in Algorithm 1. Here cases 1 and 2 are the problem and cases 3 and 4 are normal. Cases 1 & 2 are required to resolve conflicts. The HA mode resolve request(ModeConflictsResolveReq) is sent to the HA2 pod by the HA1 pod. HA2 pod resolves HA mode as shown in Algorithm 1.

The algorithm to resolve mode conflict is deciding the value of each site based on a set of parameters with the formula below where M(X) is a mode of site or cluster X as shown in Eq.1. N is the total number of variables used to decide the mode, for example, N is shown with a value of 5. It can be dynamically updated if required to add a new parameter. p1, p2, p3, p4, and p5 are constants, and values are represented by $10^{(\mathrm{N}-\text{preference value})}$. Here Preference values mean the priority of the parameter, so the highest priority will have a value as low to have a higher value.

$$M(X) = p1 * U(X) + p2 * C(X) + p3 * S(X) + p4 * T(X) + p5 * R(X) \quad (1)$$

where X is the site or cluster referred to, p1 is 10^5, p2 is 10^4, p3 is 10^3, p4 is 10^2 and p5 is 10^1. U(X) denotes user preference for the default mode to be applied for the site when mode conflict happens and possible values are {0,1}. C(X) denotes the number of Network elements connected with S-EMS. S(X) denotes Service health weightage and S(X) are expressed in Eq.2.

$$S(X) = \sum_j Ss_i * Sw_i \quad (2)$$

where Ss_i denotes i^{th} service status and the possible value for Ss_i is 1 or 0 based on whether the service is running or stopped; Sw_i denotes i^{th} service weightage and the weightage of service can be different based on criticality and importance of functions.

$$S(X) = \sum_j Ss_i = 1 \quad (3)$$

However, the sum of all service weightage is 1 as shown in Eq.3. T(X) denotes S-EMS startup time and R(X) denotes

Algorithm 1 HA Mode Conflict Resolve

Assumptions:
U(X): User preference based on network or infra setup. Possible values are {0,1}
C(X): Number of NE connected with S-EMS.
S(X): Service health
T(X): S-EMS startup time
R(X): Random number with base value 2. Possible values are {0,1}
N: Number of parameters used to make a decision, where N is 5 for example.
M(X): Mode of Site or cluster X is calculated below
$M(X) = p1 * U(X) + p2 * C(X) + p3 * S(X) + p4 * T(X) + p5 * R(X)$
Where, p1, p2, p3, p4 are constants and p1 is 10^{N}, p2 is 10^{N-1}, p3 is 10^{N-2}, p4 is 10^{N-3} and p5 is 10^{N-4}.
$S(X) = \sum_j Ss_i * Sw_i$ where ss_i is service status with 1 or 0 value based on running or stopped and SW_i is service weightage.

Output: Site(X) and Site(Y) HA mode status after conflict resolved
*I*nitialisation: Assume there are 'X' and 'Y' site
1: $valueforSiteX = 0$;
2: $valueforSiteY = 0$;
3: calculate $valueforSiteX = M(X)$
4: calculate $valueforSiteY = M(Y)$
5: ***if*** **[valueforSiteX** > *valueforSiteY]* ***then***
6: *siteX=ACTIVE*
siteY=STANDBY
7: **end if**
8: ***if*** **[valueforSiteX** < *valueforSiteY]* ***then***
9: *siteX=STANDBY*
siteY = ACTIVE
10: **end if**
11: **return**

a random number with base value 2 and possible values are 0,1}.

The HA2 pod returns HA conflict resolved values to the HA 1 pod. The HA1 pod will be confirmed with the HA2 pod if the mode value is changed and reconfirmed with a request, and both pods update mode values based on resolved conflicts, as shown in Fig.3. When the mode is changed, the respective sides perform notification requests for pods/services or applications, whichever is running in the respective cluster, to perform the necessary operations. In case cluster mode is changed from ACTIVE to STANDBY then the application or pod needs to be deleted or stopped. In case the mode is changed from STANDBY to ACTIVE then the required pods/services need to be started to provide the service. Here, the mode decision is based on the highest priority of the parameter based on the order in which it is fed to the formula. In case all parameters are the same, then the last parameter R(X) decides the value randomly. Therefore, either one of the sites will be selected as ACTIVE or STANDBY. The process is the same as mentioned above when the HA2 pod is coming up, here HA pod 2 will synchronize HA mode with other HA pod running in cluster #1, where the respective role of deciding conflicts will be done as vice-visa.

D. EMS APPLICATION HEALTH CHECK

The HA pod checks the health of S-EMS application pods, services, and related components and shares details with other HA pods through handshake functionality. The handshake interface is explained in Section III.B.

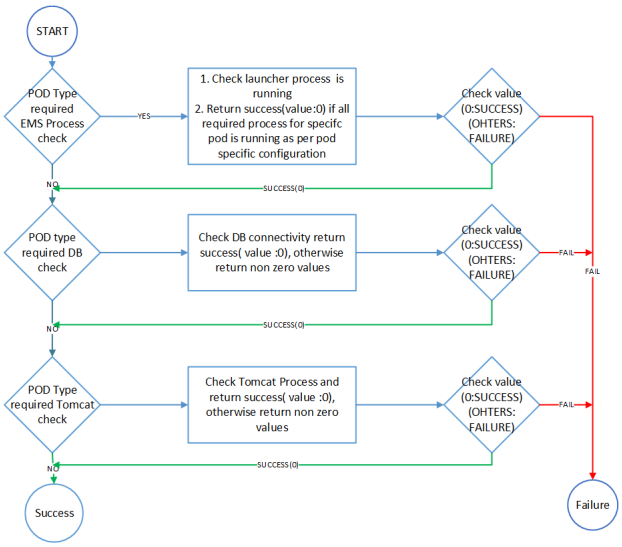

FIGURE 4. The pod health check is based on pod types.

The health of each pod may be different due to various processes or components running on each pod; therefore, checking health cannot be the same for all pods, depending on the type of pod based on the underlying process running it. As shown in Fig.4, the health of the pod is checked based on pod type. Here, the example is shown as just 3 different types of pod, but it can be more different types and the health of the pod can be done according to a specific pod based on underlying processing or services or components or library or resources being used or consumed. Based on each pod/service health, the total system health is decided. Here, the weightage of pod/service will be different from each pod/service. The total health of all pods/services is mapped to 1 and evaluated to 100%. So total health of the system is calculated as per Eq.2. There can be many pods/ services in the system. The S-EMS decides the health of the cluster based on the weightage of each pod/service and if the summation of all pod/service health is below threshold values then the system is considered abnormal and it will be considered for the geographical switch over. Switch over is explained in Section III.E. The pods/services health at the cluster level at configured intervals will be considered for the switch over.

The S-EMS performs a health check algorithm as shown in Algorithm 2. Here, S(X) denotes the pod/service health status in cluster X, D(X) denotes the threshold value to be considered for abnormality of the health of the cluster, and K(X) denotes the configured count for failure or retry before deciding on a health failure. When the health of the system fails continuously for the configured threshold D(X) for configured times K(X), the system will be considered a failure and planned to switch over. Switch-over details are shared in Section III.E.

E. SWITCH OVER

The HA pod provides a switch-over function either automatically or manually. When the Health of S-EMS is

Algorithm 2 S-EMS Health Check Decision

Assumptions:
S(X): The pod/service health in cluster X
D(X): Threshold value to be considered for abnormality of Health of the cluster.
K(X): configured count for failure or retry
$S(X) = \sum_j Ss_i * Sw_i$ where ss_i is service status with 1 or 0 value based on running or stopped and SW_i is service weightage.

Output: S-EMS Application Health Status

*I*nitialization: Assume there are 'T' Services/Pods and the retry count is a global variable
1: $threshold = 0$;
2: retryCount=0; // Global variable
3: LOOP Process
4: **for** [i = 1] to [T] **do**
5: $threshold = threshold + Ss_i^* Sw_i$
6: **end for**
7: **if** [$threshold < D(X)$] **then**
8: retryCount = retryCount +1
9: **end if**
10: **if** [$threshold >= D(X)$] **then**
11: retryCount=0
12: **end if**
13: **if** [retryCount > $K(X)$] **then**
14: retryCount = 0;
15: **return** false;
16: **end if**
17: **return true;**

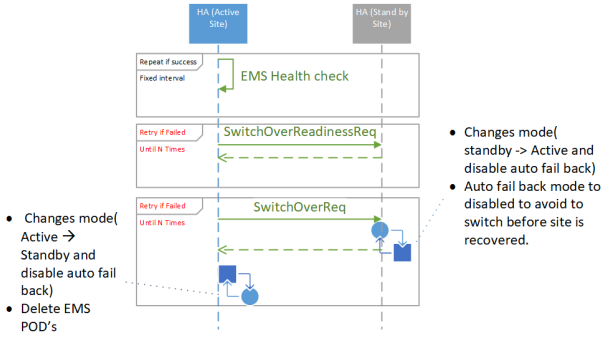

FIGURE 5. HA switch over details.

abnormal, the HA pod automatically changes the system status (ACTIVE to STANDBY). In case the user wants to change the system status manually, it provides a REST API interface for the same.

As shown in Fig.5, switch over can be triggered where the ACTIVE site HA detects an issue in the active site, and then the ACTIVE site HA pod requests the STANDBY site HA pod for the switch over request (SwitchOverReadiness-Req). The health check details are explained in Section III.E. The HA in the standby site responds to readiness for the switch over operation. The HA pod in the ACTIVE site requests switch over request(SwitchOverReq) to HA in the STANDBY site, the HA pod in the STANDBY site starts required services in the STANDBY site and the STANDBY site becomes ACTIVE at the time the previous ACTIVE site becomes STANDBY site and previously running pod/service are stopped or deleted by the HA pod. In case HA Switch over

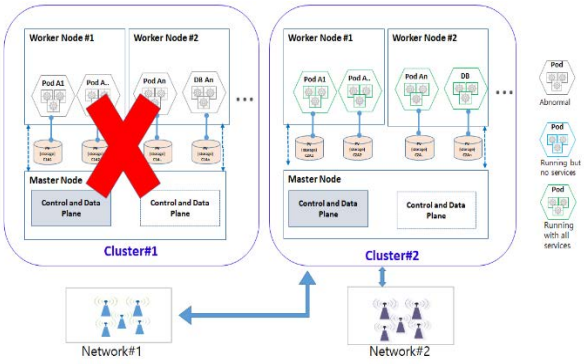

FIGURE 6. NE Management after switch over.

TABLE 2. Possible pod status in S-EMS.

Cases	S-EMS pod status	Remarks
1	Abnormal	A pod is in abnormal status
2	Running but no services	A pod is running, but no services are provided.
3	Running with all services	Running with all services and it is fully functional

Request is failed, then the system will try for a configured interval. During the switch over, the event is raised at the start and end of the switch over to know the status and progress of the user to track it later or for auditing purposes. When switching is performed, the system updates required parameters such as switch over mode, and auto fallback mode to disable and delete or start required pods/services based on-site or cluster HA mode (ACTIVE/STANDBY). Here, auto fails back mode details are explained in detail in Section III.G.

After switching over, as shown in Fig.6, pods/services run in cluster#2, Cluster#2 becomes ACTIVE, and Cluster#1 becomes STANDBY. Network elements in Network#1, Network#2... such as 4G, 5G RAN, and Core NE's in Cluster#1 is managed by Cluster#2. Fig.6 shows only two networks for simplicity and understanding; it does not mean that it will manage only two networks, the system can manage N networks based on resource capacity. Here, the pod will have the below status as shown in Table 2.

The S-EMS pod status is derived from the pod phase status. The pod phase status is status defined as per container-based platforms such as Kubernetes.

The default pod phase status is presented in Table 3. Here, an abnormal S-EMS pod status indicates that the pod is in Failed/Unknown. Running but no services mean there can be many containers, some containers related to platforms are running, but major application-related containers are not running.

TABLE 3. Pod phase definition.

Cases	S-EMS pod status	Remarks
1	Pending	After Pod is created, it's in the initial phase. Until the pod is scheduled to a node and the images of its containers are pulled and started, it remains in this phase
2	Running	At least one of the pod's containers is running.
3	Succeeded	Pods that aren't intended to run indefinitely are marked as Succeeded when all their containers complete successfully.
4	Failed	When a pod is not configured to run indefinitely and at least one of its containers terminates unsuccessfully, the pod is marked as Failed.
5	Unknown	The state of the pod is unknown because the Kubelet has stopped reporting communicating with the API server. Possibly the worker node has failed or has disconnected from the network

TABLE 4. S-EMS pod status at ACTIVE and STANDBY sites.

Type of Pod	ACTIVE Cluster	STANDBY cluster	Remarks
Application Pods	Running with all services	Running but no services	When switchover has happened status will be changed according to cluster modes such as ACTIVE or STANDBY
Platform Pods	Running with all services	Running with all services	Platform pods such as DB, File replication pods are always running, there is no change in status during the switch over

The pod can have more than one container, so simply representing it as running does not have any meaning; thus, the S-EMS defines new status called 'Running but no services'. 'Running with all services' means that all containers in the pod are running and working as expected. The pod status at active and standby sites is listed in Table 4. When switching over is performed, the S-EMS Pod status is changed based on the cluster mode (ACTIVE/ STANDBY) by the HA pod. Thus, system services are restored immediately at either site by the HA pod.

Whenever a switch-over occurs, it means that the site is a problem; therefore, an immediate switch-over again is not recommended until otherwise crashed or the issue site is recovered. Therefore, this can be avoided by adding a flag in the system as an auto failback. Details of this flag will be explained in Section III.G.

F. DATA SYNCHRONIZATION

State-dependent data synchronization between sites at real-time and scheduled intervals is required to avoid impact on the business when the switch over occurs. Data

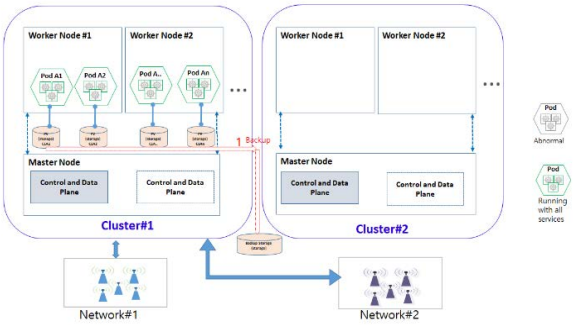

FIGURE 7. Existing ways of state-dependent data backup at ACTIVE SITE.

synchronization between clusters is the main concern; if data are stored in backup storage and when the STANDBY site becomes ACTIVE, it is required to copy data and extract data according to pod/service, which may impact the application or pod come to service. This problem is a major problem in container-based architecture where data is state-dependent and data are required to be maintained for each pod/service-wise. As shown in Fig.7, cluster #1 data is backed up to external storage at a fixed interval or in real time. The required data in cluster #1 from all pods/services are collected, compressed, and moved data to an external backup storage location using the secure file transfer protocol (SFTP) utility. This will be done frequently or in real time based on the type of data being used by the respective pods/services.

The backup data are copied to cluster #2 when cluster #2 is changed to a running state, as shown in Fig.8. As shown in Fig.8, state-dependent data are copied from the external backup to cluster #2 for each pod/service storage. It may take a lot of time to incase the size of state-dependent data is huge. The S-EMS provides a solution for how effectively data can be synchronized in real-time or scheduled time in such a way that the pod/service does not affect when it is coming up or started.

The state-dependent data of the S-EMS pods/services are synchronized in real-time or scheduled times, as shown in Fig.9. Each pod/service in cluster #1 is synchronized with the respective counter pod in cluster #2 in real-time or scheduled times. Here, cluster #2 will have all pods running but without services.

The pod in state 'running but not providing services' is explained in Section III.E. As shown in Fig.9, Cluster #2 pod will be running, but no services are provided, which means that pod will have more than one container, and some of the basic containers such as volume, DB, file replications, log related, and required for the platform are running; however, containers related to applications or services that provide response, service or function to the system or users will not be started by maintaining container status as a pending state.

When the pod is running state but not in service will have a volume attached, so data between ACTIVE and STANDBY pods can be done without any issue where pods are running,

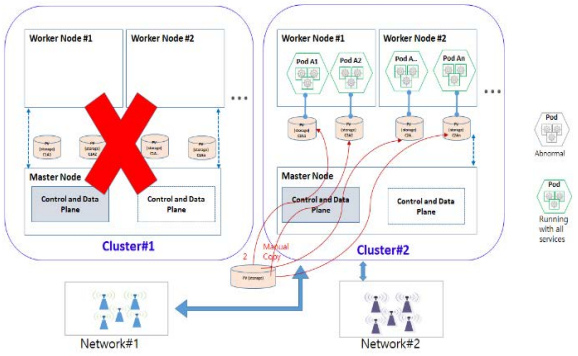

FIGURE 8. Existing ways of state-dependent data restore at cluster when it becomes ACTIVE.

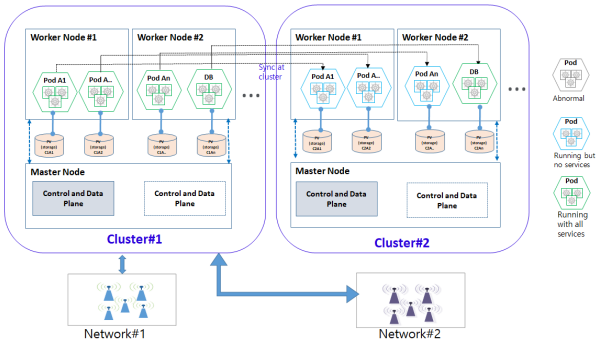

FIGURE 9. Existing ways of state-dependent data backup at ACTIVE SITE.

so respective storage or volumes are accessible to respective pods. The pod is running but no services are shown in Fig.10. As shown in Fig.10, the pod has some set of basic or initialization containers that will be running, and the remaining containers related to applications or services are pending based on the cluster mode. If cluster mode is in an ACTIVE state, then the pod is running and all services will be started, if the cluster is in STANDBY mode then only initialize containers or basic containers are started and other containers are not started. Using this logic, S-EMS provides data synchronization between the ACTIVE and STANDBY sites. When the ACTIVE site is abnormal, the STANDBY site becomes ACTIVE as explained in the switch over concept in Section III.E. At that time, state-dependent data are already synchronized to the standby site, so no further action is required at the STANDBY site, just the pod can be running with all services state as shown in Fig.9.

The S-EMS provides state-dependent data synchronization between clusters by respective pods storage directory where pods are running in all services in the ACTIVE site and STANDBY site pods in 'running but no Service' status. The pod status is changed based on the cluster or site mode (ACTIVE or STANDBY). There is no extra back or storage location required, and the time required to extract while the service is coming is not required, so it reduces storage and improves service availability.

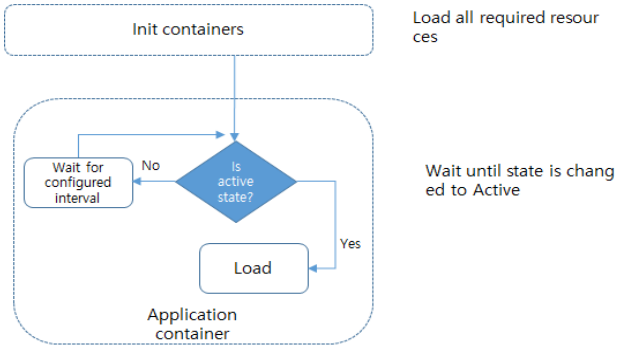

FIGURE 10. S-EMS pod is running, but No service status logic.

TABLE 5. Possible cases after the site failed or abnormal.

Cases	Cluster #1 Status	Cluster #2 Status	Remarks
1	NORMAL	NORMAL	Here both sites are NORMAL, so there are two possible a. The active site can be maintained b. Switching back to the old site is possible.
2	NORMAL	ABNORMAL	Cluster #1 will be an active site.
3	ABNORMAL	NORMAL	Cluster #2 will be an active site
4	ABNORMAL	ABNORMAL	Not valid case

G. AUTO FAILBACK

The S-EMS HA pod supports an auto-failback option that provides control over when a site or cluster is crushed. The system will switch to another site, and should not be allowed back to avoid the split-brain issue. When a site or cluster fails or is in abnormal, the user may be required to recover the site manually; because the cluster will have many components, resources, networks, and configurations. The cluster not only has an S-EMS application, but can also have many other applications, services, or utilities running over many namespaces. Therefore, frequent switching between sites is not recommended until otherwise required. So auto failback is controllable in S-EMS to avoid unwanted switch over irrespective of cases such as crash or disaster or failure is recovered or not recovered site. The possible cases are listed in Table 5.

In case of the failed site is recovered, it does not require to change to the original site or a switch over as services are running on the backup site. In that case, auto failback is not required until the otherwise current running site has the issue of supporting services or failure. As shown in Table 6, the control auto falls back after the switch over is controlled by the auto failback flag in the S-EMS. The auto failback flag is

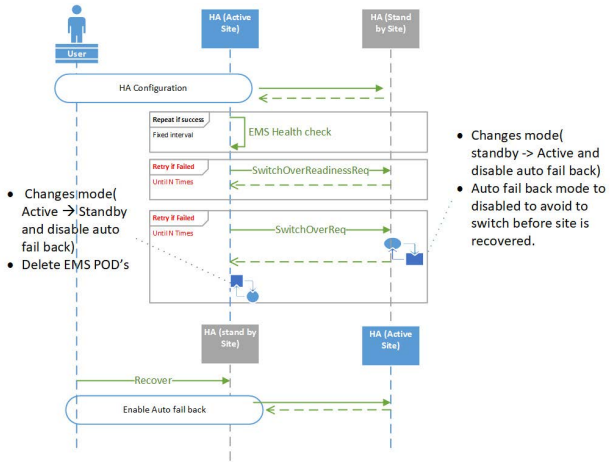

FIGURE 11. S-EMS auto failback configuration support.

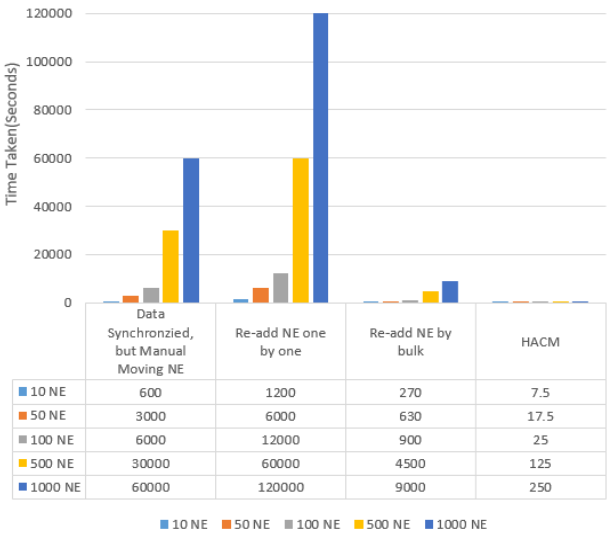

FIGURE 12. HACM evaluation report.

TABLE 6. Possible values for auto failback flag.

Cases	Auto failback Flag Value	Remarks
1	DISABLE	Auto failback is disabled, there will not be any switch-over allowed.
2	ENABLE	Auto fails back is allowed when there is a failure in the current running site and the other cluster or site is normal and healthy.
3	ENABLE_SWITCH_IMMEDIATE	Auto rollback is done immediately after the flag is enabled if the destination site is normal and healthy and the user sets the preferred site as the primary site. If the primary site is not set, the then-current state is maintained, and switching over will not be done.

reset whenever the switch is over being triggered, so further failback is not allowed until otherwise the user is confirmed that crashed site has recovered and made user preference whether failing back is required immediately or following failure.

As shown in Fig.11, S-EMS disables the auto rollback flag every time the switch-over occurs. When a user is recovering failed site or cluster, the user can change the options that need to be done according to the user's choice. The possible options for auto failback are listed in Table 6. As explained in Table 6, the DISABLE flag will not allow switching to other sites. The ENABLE flag will allow switching to another cluster or site if the other cluster or site is in the normal state and healthy.

ENABLE_SWITCH_IMMEDIATE makes a switch over if the destination site is in normal, healthy condition and the destination is configured as a primary preferred site by the user. Thus, different possible options can be used per user preference, based on their network setup or configurations.

IV. PERFORMANCE EVALUATION

The HACM performance evaluation result is performed with various criteria such as switch-over time of NE, backup storage required estimation and service start-up time. Each category is used to determine a different key performance indicator (KPI), such as the time required to start service after the switch over, and the storage required in terms of memory, network resource assignment, or movement during switchover or after the switchover. Details of each major category are described in the following sections.

A. SWITCH OVER PERFORMANCE EVALUATION

The HACM performance is evaluated under various conditions, such as data synchronized but manually moving NE between clusters, reading NE one by one manually, and re-adding NE bulk after switching over.

The details of the performance comparison result are shown in Fig.12. Data synchronized but manually moving NE after the switch over requires considerable manual effort, and the time taken to bring service to NE is greater. It increases promotionally when the number of NE increases. This may be an issue in the presence of many NE. In the case of 'Re-add NE one by one' option is performing worse than compared with the case 'Data synchronized but manually moving NE'. However, this requires more time and effort. The case 'Re-add by NE bulk' options provides better results compared with the earlier two options; however, the time taken is quite high when the number of network elements is greater. The HACM provides constant results when the number of NE increases as each NE is synchronized automatically with the S-EMS in the background without any manual work and effort. We observed that a constant time was taken or almost very less time was taken to bring into services. The assumption is that manual movement is dependent on user working speed, so considered as average users speed is 1 minute for manually moving the NE and 5 minutes for creating a new NE, as it has many parameters to configure by considering the large number of NE.

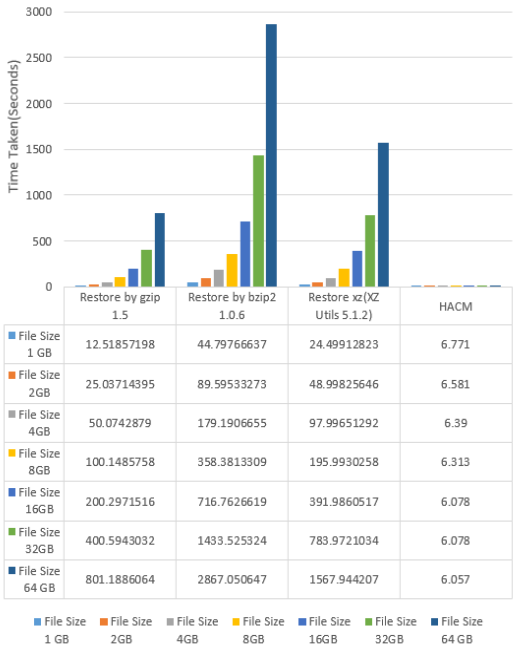

FIGURE 13. HACM performance evaluation report.

B. SERVICE START-UP EVALUATION

The HACM storage is evaluated with an external backup method where state-dependent data decompress is evaluated with various sizes from 1 gigabyte (GB), 2 GB, 4GB, 8 GB, 16 GB, and larger sizes with various extraction tools such as gzip 1.5, bzip2 1.06, and XZ utils 5.1.2 version used. Our simulator was run on 2 bare-metal servers composing 80CPU and 500GB memory each and a dedicated connection between servers with a 1gbps Ethernet interface. The observation is the same for all other methods in that the time taken to extract data increases when the backup data size is increased. When the size of the back is 64 GB, it is taking almost approximately 13 minutes (801 seconds) by gzip to extract data. The details of this comparison are shown in Fig.13. The main observation is that a smaller data backup size can be handled within a limited service downtime. Most Tier 1 network operators allow a maximum of 5 minutes, so less than 8GB of backup data can be handled with other methods; when the state-dependent data size is more than 8GB or very large, it is not possible to handle with the traditional method to extract data when the site is becoming active, which leads to loss of functionality or services can't be started until the restoration of data is applied when switch over occurs. The HAMC required a very short time of approximately 6 to 7 seconds. The reason for taking less time is that HAMC performs incremental restoration continuously; only for the last interval data may not be extracted or during extraction there can be a chance to switch over, so the size of backup data is not more than 1 GB for all cases, so it helps to start services immediately within 7 seconds.

Here, we assumed that the transfer data between sites is constant, and it is the same for all approaches. So only the extraction part is considered as transfer data is happening in the background, so it is not worth considering for evaluation, so it is not considered for all cases mentioned in the evaluation report shown in Fig.13. The HACM recommends the user that even if very large state-dependent data are available, as service start time is not impacted as data is being updated in the background, there may be very little or no data required to extract when the service is being started.

TABLE 7. Storage Usage Estimation comparisons.

Solution Type	Active Site storage in GB	External Back up Storage in GB	Backup Site Storage in GB
Existing solutions	N	N	N
HACM	N	-	N

C. BACK-UP STORAGE ESTIMATION EVALUATION

The HACM storage is evaluated with the storage required to support the above-said functionalities using the traditional approach. Assumed that N is the required storage size in GB.

As explained in Table 7, existing prior art solutions are required 3N storage to support the solution; however, HACM requires only 2N, which is a 33.33% storage reduction, and there is no extra manual effort required to handle storage. The extra storage used in the existing solutions or the traditional approach is shown in Fig.7 and 8. The HACM reduces data transfer by 50% as transfer between the active site to backup storage and backup storage to the standby site; thus, network and bandwidth usage is reduced by 50%.

V. CONCLUSION

In this paper, we discussed the feasibility of applying HACM which enables HA for state-dependent data and services to be intrinsically supported by the underlying application running over between independent clusters in different geographic locations. HACM provides automatic switch over between clusters based on the health of application or pod/service by handshake between HA pods running in independent sites, provides switch over option to user to make preference to configure site when a site is recovered after the crash to decide whether the running site needs to be in active more or standby mode, state-dependent data synchronization between clusters without extra storage with less network bandwidth usage by avoiding data transfer between active to backup site and backup site to standby site, controlling auto failback flag to provide more flexibility and user convenience based on demand or priority, resolve systematic way if there are conflicts in cluster mode such as active-active or standby-standby and having internal handshake to support the above-said functionalities and having pod with running, but no services, pod with running status and provides all services state between clusters based on the mode of cluster and controls

pods/services or state of cluster to support the above-said functionalities.

The HACM performance result is simulated with other existing approaches for HA switch over possible options available and used by Tier 1 network operators in network industries, where state-dependent data need to be maintained and stored independently to respective services, applications, or pods. HACM provides more benefits in terms of storage reduction, and manual intervention of operations.

Towards future work, we will extend HACM and further improve KPIs by introducing to considering the above-said functionalities with multiple clusters with N: K redundancy, where N represents currently active sides or clusters and K represents backup clusters or sites. The current solution is focused on 1:1 where N and K are 1 and data is state-dependent and further research is required when N and K are configured dynamically.

REFERENCES

[1] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*. Cham, Switzerland: Springer, 2017, pp. 195–216.

[2] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Kubernetes as an availability manager for microservice applications," 2019, *arXiv:1901.04946*.

[3] G. Chandrasekaran, B. Ramasamy, P. Dhondi, P. B. Thorat, and R. Challa, "CASE: A context-aware storage placement and retrieval ecosystem," *IEEE Access*, vol. 10, pp. 1956–1967, 2022.

[4] Nokia. *Blog Containers and the Evolving 5G Cloud-Native Journey*. Accessed: 2022. [Online]. Available: https://www.nokia.com/blog/containers-and-the-evolving-5g-cloud-native-journey/

[5] *The 3rd Generation Partnership Project, 3GPP Standard Release 15 and Release 17*. Accessed: 2022. [Online]. Available: https://www.3gpp.org/release-15 and https://www.3gpp.org/release-17

[6] ETSI. *European Telecommunications Standards Institute*. Accessed: 2022. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/040/04.01.01_60/gs_NFV-IFA040v040101p.pdf

[7] *European Telecommunications Standards Institute, ETSI Release-4*. Accessed: 2022. [Online]. Available: https://docbox.etsi.org/ISG/NFV/Open/Other/ReleaseDocumentation/NFV(21)000025_NFV_Release_4_Definition_v0_3_0.pdf

[8] *Samsung 5G Core White Paper, Cloud Native 5G Core*. Accessed: 2022. [Online]. Available: https://images.samsung.com/is/content/samsung/p5/global/business/networks/insights/white-paper/cloud-native-5g-core/Samsung-5G-Core-Vol-2-Cloud-Native-5G-Core.pdf

[9] R. Morabito, "Virtualization on edge devices with container technologies: A performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.

[10] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May 2016.

[11] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 939–951, Mar. 2021.

[12] D.-H. Luong, H.-T. Thieu, A. Outtagarts, and Y. Ghamri-Doudane, "Predictive autoscaling orchestration for cloud-native telecom microservices," in *Proc. IEEE 5G World Forum (5GWF)*, Jul. 2018, pp. 153–158.

[13] D. K. Limbu, A. M. Connor, and S. G. MacDonell, "A framework for contextual information retrieval from the WWW," Auckland Univ. Technol., Auckland, New Zealand, Tech. Rep., 2005. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1407/1407.6100.pdf

[14] A. Mseddi, W. Jaafar, H. Elbiaze, and W. Ajib, "Joint container placement and task provisioning in dynamic fog computing," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 10028–10040, Dec. 2019.

BOOPATHI RAMASAMY received the B.E. degree in computer science and engineering from Bharathiar University, Tamil Nadu, India, in 2004. From 2004 to 2007, he worked on an embedded system at Flextronics, Bengaluru, India. From 2007 to 2013, he worked on shaping various telecom network products at Huawei Technologies India Pvt. Ltd., Bengaluru. Since 2014, he has been a Principal Engineer with Samsung Electronics, HQ, South Korea, and plays the role of Security System Architect. His research interests include 5G networking, information-centric networking, network resource management, cloud, micro-service design, and cyber security.

YEONJOO NA received the B.E. degree, in 1992, and the M.S. degree in industrial engineering from Hongik University, Seoul, South Korea, in 1994. From 1994 to 1996, he was at Daewoo Information System, South Korea. Since 1996, he has been a Principal Engineer with Samsung Electronics, South Korea, and plays an EMS System Architect role. His research interests include 5G network management, data-lake massive data management, and virtualized system management.

WEONSU KIM received the B.E. degree, in 1993, and the M.S. degree in electronic engineering from Kyungpook National University, Deagu, South Korea, in 1995. Since 1995, he has been a Principal Engineer with Samsung Electronics, South Korea. He plays the Architect role of the network management system. His research interests include 5G network management, private network management, and virtualized system management.

KYOUNGBEOM CHEA received the B.E. degree in electrical engineering, in 2009, and the M.S. degree in digital media communication from SungKyunKwan University (SKKU), South Korea, in 2018. From 2009 to 2016, he was a Developer in the network management system domain at Samsung Electronics, South Korea, where he was a Senior Engineer in the network function virtualization domain, from 2018 to 2022. Since 2022, he has been working with Samsung Electronics and plays network product strategy. His research interests include 5G networking, network function virtualization, and Open RAN.

JUN KIM received the B.E. degree in computer engineering from Chosun University, Gwang, South Korea, in 2007. Since 2007, he has been a Senior Research Engineer with Samsung Electronics, South Korea. His research interests include 5G networks, network function virtualization, cloud, and micro-service design.