B5G: Predictive Container Auto-Scaling for Cellular Evolved Packet Core

YAHUZA BELLO1, ALAA AWAD ABDELLATIF2, MHD SARIA AL-LAHHAM2, AHMED REFAEY HUSSEIN3, AIMAN ERBAD4, AMR MOHAMED2, and MOHSEN GUIZANI2

1Manhattan College, Riverdale, New York, USA.
2College of Engineering, Qatar University, Doha, Qatar.
3Western University, London, Ontario, Canada.
4College of Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar.

Corresponding author: Aiman Erbad (e-mail: aerbad@hbku.edu.qa).

This work was made possible by NPRP grant # NPRP12S-0305-190231 from the Qatar National Research Fund (a member of Qatar Foundation).

ABSTRACT In order to maintain a satisfactory performance in the midst of rapid growth of mobile traffic, the mobile network infrastructure needs to be scaled. Thus there has been significant interest in scalability of mobile core networks and a variety of scaling solutions have been proposed that rely on horizontal scaling or vertical scaling. These solutions handle the scaling of the mobile core networks’ elements on virtual machines (which normally take at while to create) with the help of customized modules at the cost of increased overheads. Utilizing Amazon Web Services (AWS) embedded features, we present two predictive horizontal auto-scalers for containerized and non-containerized versions of EPC that scales the two versions of the EPC according to their respective CPU utilization. Additionally, we propose an efficient task assignment scheme for AWS that aims to maximize throughput and achieve fairness among competing instances. In particular, we propose two solutions: Relaxed Optimized Solution (ROS) and a Heuristic Approach (HA). Leveraging AWS environment, we implemented and evaluated the two proposed auto-scaling models based on the attachment success rate, latency, CPU usage and RAM usage. Our findings show the superiority of container-based model over VM-based model in terms of resource utilization. The obtained results for the two proposed task assignment solutions demonstrates a significant improvement both in fairness and throughput compared to other existing solutions.

INDEX TERMS Evolved Packet Core, Amazon Web Services, Scaling, Auto Scaling Group, Optimization, Container, Virtual Machine, Implementation

I. INTRODUCTION

The past decade has seen mobile operators experience a dramatic rise in the use of hand-held devices, along with a corresponding surge in network traffic. By the end of 2022, CISCO’s annual Internet report predicts that mobile data will tally up to 77.5 exabytes monthly, a seven-fold increase from 2017 [1]. To meet the client’s Quality of Service (QoS) demands and to handle the surge in workload affecting their infrastructure, mobile operators have been researching better solutions to address the enormous increase in mobile data traffic.

Traditionally, Long Term Evolution (LTE) networks have been proprietary hardware/software components provided on-sites by vendors. It is therefore extremely difficult for mobile operators to upgrade or scale them. In order to scale the EPC, for example, a higher capacity entity will be needed to replace the current one. The operator won’t only incur higher Capital Expenditures (CAPEX), but also higher Operational Expenditures (OPEX). As mobile operators are bound to specific vendors, they also face compatibility issues.

In recent years, cloud computing technologies [2] and Network Function Virtualization (NFV) [3] have rendered EPC entities virtualized and deployed on servers of Cloud-based environments like Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP). This solution eliminates the proprietary hardware dependences of the mobile operator and promises a greater return on both CAPEX and OPEX. In addition, the EPC entities deployed on the cloud environment will not only result in high available network services, but also high scalable EPC. It is therefore
imperative for the cloud deployment of EPC entities to have an excellent scaling strategy that adopts horizontal or vertical scaling or a combination of both.

A number of research publications evaluated horizontal (adding or removing instances of an entity) and vertical (to increase or decrease the amount of resources allocated to entities) EPC scaling strategies in order to propose new solutions for a potent, scalable EPC. A number of studies point to solutions in which either all the EPC entities [4], [5] or a few of the entities [6]–[10] can be horizontally scaled. Researchers have also attempted to scale EPC entities horizontally and vertically [11]–[13]. In [13], a method is proposed in which EPC components are directly scaled both horizontally and vertically. The authors presented a threshold-based scaling strategy for determining whether to scale horizontally or vertically based on the variations in workload of the EPC entities. This strategy is composed of three components: a data collection component for monitoring and collecting relevant performance data, a decision component that determines if the system should be scaled horizontally or vertically based on the data recorded from the first component and an execution component that is responsible for starting the scaling process. Even though the authors implemented their scaling strategy on AWS, they do not fully capitalize on the services of AWS like auto-scalers and load balancers. Furthermore, they run the EPC entities on Virtual Machines (VMs), which usually take longer to start when performing horizontal scaling. By utilizing AWS’s features or services, we will not only be able to scale more efficiently, but we can also minimize the overheads incurred from the scaling components, which has been shown in our preliminary implementation in [14].

Considering this, we present in this paper two predictive auto-scalers for containerized and non-containerized versions of EPC, which both leverage the embedded features (i.e., auto-scalers and load balancers) provided by AWS. In particular, AWS’s Elastic Container Service (ECS) is used to run containerized version of the EPC, and the AWS’s auto-scaler is utilized to scale in/out the EPC entities based on their workload capacities, via an Elastic Load Balancer (ELB). In a similar fashion, AWS’s EC2 instances are used to host the non-containerized versions of the EPC and AWS’s auto-scaler is again utilized to scale in/out the EPC entities based on their workload capacities maintained by an ELB.

Although the proposed models are implemented as a use case for EPC, it will be applicable to 5GC and Beyond 5G (B5G), as both core networks are likely to share almost similar architectural patterns [15]. We further formulate an efficient task assignment scheme for AWS to achieve proportional fairness and at the same time maximize throughput among competing instances. We solve the formulated problem using two approaches: Relaxed Optimized Solution (ROS) and a Heuristic Approach (HA). The breakdown of this paper’s contributions is described below:

- Utilizing the AWS auto-scaling and load balancing services, we implement a container-based predictive auto-scaler, which is based on a lightweight version of the containerized EPC entities on AWS, in addition to a predictive auto-scaler for the VM-based EPC implementation. AWS’s ECS provided us with the option to host containerized EPC entities and aggregated the containers to the Auto Scaling Group (ASG) and ELB at ECS level. Every time the auto-scaler’s policy (which is based on CPU usage) is breached, the auto-scaling feature is triggered.

- We present a predictive auto-scaler for non-containerized versions of EPC entities using AWS’s auto-scaler and load balancer. The EPC entities are ran on EC2 instances and attached to the ASG and an ELB. Every time the auto-scaler’s policy (which is based on CPU usage) is breached, the auto-scaling feature is triggered.

- We formulate an efficient task assignment scheme that maximizes throughput while maintaining proportional fairness among multiple instances. Then, ROS and HA approaches are proposed to solve the task assignment problem, which can be used for scaling the EPC entities horizontally and/or vertically.

- We perform a series of experimental analysis to demonstrate how auto-scaling affects resource utilization of EPC entities, while comparing between the container-based and VM-based implementations of the EPC entities (both containerized and non-containerized versions). For both proposed models, our findings indicate that memory usage remains minimal as workload increases while CPU usage saturates as more requests are handled. Furthermore, the ROS and HA solutions were evaluated and compared with the famous Round Robin (RR) algorithm.

The rest of the paper is organized as follows: in section II, we put forward the relevant knowledge on virtualized EPC, scalability mechanisms, virtualization technologies and the relevant literature research findings. We describe in detail the proposed container-based and VM-based EPC implementations on AWS in Section III. Section IV presents the proposed task assignment scheme. We present a detailed explanation of our optimization model. A description of the Relaxed Optimized Solution (ROS) and Heuristic Approach (HA) concludes the section. Section V presents the performance assessment of the auto-scalers and the solutions for task assignment. We conclude our paper in section VI with a discussion of potential future research.

II. BACKGROUND AND RELATED WORK

The purpose of this section is to present the basic background knowledge and related literature works that have been reported in the literature regarding virtualized EPC, scalability mechanisms, and virtualization technologies.

A. VIRTUAL EVOLVED PACKET CORE

Radio access networks and EPCs make up legacy LTE systems. EPC is comprised of Mobility Management Entity
(MME), Home Subscriber Server (HSS), Serving Gateway (SGW) and Packet Data Network Gateway (PGW) [16]. The EPC, which is driven by these components, supports the traffic flowing from the User Equipments (UEs) to the eNodeBs. The HSS is a database used as a storage for UEs’ data, including their authentication keys. In coordination with HSS, the MME is responsible for handling UE authentication, configurations such as that of default bearer, and mobility management duties like paging and handover of UEs. Inbound traffic is routed through the SGW while outbound traffic is routed through the PGW. Further, the SGW configures the uplink and downlink for data transmission, and the EPC traffic is routed to the Internet via PGW.

The NFV technology allows EPC entities to be virtualized and become highly scalable through the use of virtual functions (VNFs). Network Operators can rely on virtual EPC (vEPC) for scalability and high available network services during peak hours to meet the ever-increasing network traffic.

The literature offers a variety of novel solutions for vEPC ranging from partialized vEPC (which partly virtualize the EPC) to full vEPC (which fully virtualize the EPC) [17], [18]. Partially virtualized EPC is one of the most widely adopted techniques since the SGW and PGW are decoupled into control and data plane entities (i.e., SGW-C and PGW-C, and SGW-U and PGW-U) that can be implemented using virtualization and hardware appliances to enable control signaling and data traffic handling, respectively [19]. We choose to use OpenAirInterface (OAI) Open Source Project’s complete vEPC implementation in this project.

### B. SCALING STRATEGIES

As the name suggests, scalability refers to a system’s ability to adapt to variations in workloads and act appropriately (to add or remove appropriate components). There are two main types of scaling: horizontal, which is scale in or out, and vertical, which is scale up or down [20]. When horizontal scaling is performed, one or more component is attached or detached from a network (more or less virtual machines). Contrary, the vertical scaling process involves increasing or decreasing the resource allocation to the instance (i.e., adding or removing more storage or CPU).

Mobile operators can dynamically scale mobile EPC entities horizontally or vertically to meet changing workloads and optimize network performance by horizontally or vertically scaling the virtualized components. Thus, numerous studies investigated and proposed strategies to scale EPC entities both horizontally and vertically. In [5], the authors discussed two conceptual models for vertically scaling of EPC entities. In the first model, the EPC entities are deployed on virtual machines (VM) that allows for vertical scaling, while the second model adopts a distributed approach similar to [6]. Each individual EPC entity is divided into front-end model, workers and a database to leverage the functionality of horizontal scaling. In [12], the authors presented an elastic scaling scheme for a cloud-based 5G system for both horizontal and vertical scalability, based on thresholds. According to the CPU usage, RAM usage and mean opinion score, a decision module initiates the scaling process (horizontally or vertically).

### C. VIRTUALIZATION TECHNOLOGIES

Virtualization has become a core concept driving cloud computing and thus has seen a tremendous research effort from both academia and industries. Virtualization technologies aim to divide physical servers into multiple separate platforms to run several operating systems (OS) utilizing the resources available on the host (i.e., the physical server) [23]. It does so by adding an abstraction layer (usually called hypervisor or Virtual Machine Monitor (VMM)) above the host’s hardware resource or operating system. With this technology, several execution environments (in the form of VMs or containers) can be deployed in a single physical server without any interference from neighbouring VMs or containers and at the same time share the host resources. Thus having an optimal resource utilization of the host server. This form of virtualization is adopted in VMs.

Another most widely adopted form of virtualization is container-based virtualization. It has gained a significant interest from both academia and industries because it can be utilized to deploy software applications in cloud-based environments. Virtualization based on containers does not require an abstraction layer such as a hypervisor, as opposed to virtualization based on VMs. The virtualization layer runs as
an application (usually called container runtime) on top of the host OS’s kernel and allows it to co-host multiple containers simultaneously. This is achieved by utilizing namespaces and control groups features of the OS kernel [24].

Table 1 presents a comparison of the benefits of using container-based implementation over the conventional VM-based implementation in terms of portability, scalability, application management, etc.

III. EPC IMPLEMENTATIONS ON AWS

In this section, we describe the two proposed implementations. We present the container-based implementation and describes its architecture within AWS followed by a detail description of the VM-based implementation within AWS as well.

A. CONTAINER-BASED IMPLEMENTATION

As shown in Fig 1, the proposed container-based predictive auto-scaler is based on a lightweight version of the EPC entities (which are Docker images) deployed on Amazon Web Services and a simulated RAN. AWS runs the containers for these LTE EPC entities via its built-in ECS as depicted in Fig. 1. Since the entities are Docker-based images, they are more scalable than VMs. The container-based entities are specifically deployed on container instances in a private subnet. By leveraging ECS’s service feature, each EPC entity has its replicas, which are attached to the ASG at the ECS level. Due to its close integration with the AWS, ECS can be easily managed via AWS embedded ECS service feature. It is important to note that the integration of ELB with ECS occurs at the container level (i.e., every task is automatically included in the ELB’s target group).

As illustrated in Fig. 1, the ELB is internal to our virtual private cloud (VPC) (located across two subnets and in two Availability Zones(AZ)) and cannot be accessed publicly since it is inside an enclosed subnet and a private zone. Containers MME and SPGW-U are attached to both ELBs and ASGs (each container has a unique ASG with a unique auto scaling policy). Containers HSS-Cassandra (which contains the HSS container and the Cassandra database container) and SPGW-C are both only attached to ASGs uniquely with a unique auto scaling policy. Radio access traffic is routed to the MME container and SPGW-U container via the ELB. A violation of their auto scaling policy results in horizontal scaling of these entities. Other containers will similarly be scaled horizontally whenever their auto scaling policies are violated.

While a personal load balancer may be less expensive to set up, it may require specialized skills to setup a load balancer that will be efficient and highly available. In contrast, AWS ELB is part of the free-tier services of AWS (except for network load balancer). All load balancers and auto-scaling capabilities offered as a service by AWS are guaranteed to be in working condition and have a high level of availability. Thus, both services are frequently updated as requirements for the new emerging technology, such as high scalability and increase in requests of load balancing. Configuration-wise, AWS provides a few knobs for configuring the load balancer and ASG according to desired functionality.

B. VM-BASED IMPLEMENTATION

The proposed predictive auto-scaler for the VM-based EPC implementation is shown in Fig. 1. Unlike the container-based implementation, each EPC entity is hosted in a VM
Containers run on the OS and as such share the OS kernel.

Virtual machines virtualize a server and run its own OS.

Containers are highly portable because they pack microservices with the entire dependencies including OS and libraries required in a single image, which can be run on a container runtime on any server.

Virtual machines are not as portable as containers because the applications within a virtual machine are highly dependent on the OS as well as the emulated hardware.

Applications run on virtual machines are harder to maintain due to the absence of microservice-based architecture. Updating or troubleshooting part of the application will require the disruption of the whole application.

Applications are highly scalable using many of the container orchestration platforms such as Docker Swarm and Kubernetes.

Virtual machines are less scalable than containers as they are mostly in gigabytes size, which makes scalability a bit hard.

Containers have a very fast instantiation time within the range of seconds.

Virtual machines have instantiation time in the range of minutes.

Containers provide light isolation from the host OS, which results in a light security boundary that can be easily exploited by hackers.

Virtual machines provide high isolation between neighbouring system, which results in a hard security boundary.

| Feature          | Containers                                           | Virtual Machines                                              |
|------------------|------------------------------------------------------|---------------------------------------------------------------|
| Operating system (OS) | Containers run on the OS and as such share the OS kernel. | Virtual machines virtualize a server and run its own OS.       |
| Portability      | Containers are highly portable because they pack microservices with the entire dependencies including OS and libraries required in a single image, which can be run on a container runtime on any server. | Virtual machines are not as portable as containers because the applications within a virtual machine are highly dependent on the OS as well as the emulated hardware. |
| Application management | Container-based applications are easy to maintain as they adopt microservice-based architecture. Each container can be updated and maintain individually. | Applications run on virtual machines are harder to maintain due to the absence of microservice-based architecture. Updating or troubleshooting part of the application will require the disruption of the whole application. |
| Scalability      | Containers are highly scalable using many of the container orchestration platforms such as Docker Swarm and Kubernetes. | Virtual machines are less scalable than containers as they are mostly in gigabytes size, which makes scalability a bit hard. |
| Instantiation time | Containers has a very fast instantiation time within the range of seconds. | Virtual machines has instantiation time in the range of minutes. |
| Security boundary | Containers provide light isolation from the host OS, which results in a light security boundary that can be easily exploited by hackers. | Virtual machines provide high isolation between neighbouring system, which results in a hard security boundary. |

IV. AWS TASK ASSIGNMENT

This section proposes an efficient task assignment scheme in AWS. The purpose of this scheme is to assign distinctive tasks or requests to the available instances efficiently and fairly, while considering diverse instances capabilities in terms of computational resources such as RAMs, CPU processing speed, etc. The objective of our task assignment scheme is twofold: (i) maximizing the overall throughput of the AWS in terms of successful task executions within a given time-frame; (ii) maintaining fairness among the various instances, i.e., all instances should be allocated tasks in a fair manner.

A. PROBLEM FORMULATION

In what follows, we assume that every instance \( i = 1, \ldots, I \) has been given a set of tasks \( n_i \) \( (n_i = 1, \ldots, N) \) within a time-frame \( T \). Hence, the overall throughput of our system is written as:

\[
\eta = \sum_{i=1}^{I} \frac{n_i}{T_i},
\]

where \( T_i \) is the execution time of instance \( i \) to finalize the processing of the assigned \( n_i \) tasks. This execution time depends on two main factors, i.e., sojourn time and computation time.

Herein, the sojourn time presents the total waiting time an instance is expected to take to process the task. We opt to leverage the queuing model to define the sojourn time. Specifically, we calculate the sojourn time using the famous M/M/1 queuing model \([25], [26]\). Hence, for the assigned tasks with same priority, the average sojourn time at instance \( i \) is written as

\[
S_i = \frac{1}{\mu_i - \lambda_i} = \frac{1}{\frac{\mu_i}{T} - \frac{\lambda_i}{T}},
\]

where \( \lambda_i \) is the arrival rate of the tasks at instance \( i \), and \( \mu_i \) is the service rate (or tasks processing rate). It is assumed that all assigned tasks will be temporarily stored in the
instance’s buffer, while neglecting the buffer overflow since it is assumed that $\lambda_i < \mu_i$.

For the computation time per instance $i$, it depends on the amount of CPU cycles required to carry out a task $n_i$, which is denoted by $\psi_{n_i}$. Thus, the amount of CPU cycles required at instance $i$ to run the assigned tasks is ($\sum n_i\psi_{n_i}$). Hence, the computation time for executing the allocated tasks at instance $i$ is given by

$$T^c_i = \frac{\sum n_i\psi_{n_i}}{f_i\cdot\delta_i},$$

(3)

where $f_i$ is the CPU-cycle frequency of instance $i$, and $\delta_i \in [0 \cdot 1]$ is the processing factor that represents the dynamic processing capability of instance $i$ with time (in order to consider the effect of adding/removing some computational capability to/from any instance). Accordingly, the execution time of instance $i$ to finalize the assigned tasks is defined as

$$T_i = T^c_i + S_i.$$  

(4)

For maximizing the throughput in AWS while maintaining the fairness, we define our objective, using the concept of proportional fairness [27], as follows:

$$U = \sum_{i=0}^{I} P_i \cdot \log \left( \frac{n_i}{T_i} \right),$$

(5)

where $P_i$ is the assignment probability of instance $i$ (i.e., the likelihood of getting a new task assignment), that is written as:

$$P_i = \delta_i - \frac{1}{N} \sum_n n_i(t),$$

(6)

where $N$ is the total number of tasks. This assignment probability plays an important role to achieve the fairness between different instances, since it decreases for the instance that is assigned more tasks recently (in order to guarantee maintaining a fair allocation on average for all instances, even when adding/removing an instance). Thus, to obtain the optimal assignment strategy of tasks to different instances that maximizes the throughput while maintaining the fairness, our optimization problem is formulated as:

$$\max_{n_i} U$$

(7)

s.t. $\sum n_i = N$  

(8)

$0 \leq n_i \leq N$  

(9)

The constraint in (8) warrants that all the tasks are allocated to the available instances, while the constraint in (9) sets boundaries to the number of tasks that can be allocated to any instance. The optimization variables in (7) are the $n_i$’s, i.e., we need to assign the upcoming tasks to the available instances such that the total throughput is maximized while maintaining the fairness between all instances. The formulation in (7) is a non-linear integer program, which is NP-complete problem [28]. The well-known methods of solving such a problem using convex optimization or a Geometric Program tool could not be feasible because of constraint (8) and the non-linearity of the objective function. The following are two approaches we propose to solve these problems.

B. THE PROPOSED TASK ASSIGNMENT SOLUTIONS

To solve the formulated problem in (7), we propose two solutions: relaxed optimized solution and heuristic approach.

1) Relaxed Optimized Solution (ROS)

This solution considers the case of a low-load system, where the rate of arrival tasks is much less than the processing rate of the available instances. In this case, by substituting with (2) and (3) in (5), the objective function can be written as

$$U = \sum_{i=0}^{I} P_i \cdot \log \left( \frac{n_i(T - n_i \cdot C_i)}{n_i \cdot C_i(T - n_i \cdot C_i) + T \cdot C_i} \right),$$

(10)

where $C_i = \frac{\psi_{n_i}}{f_i \cdot \delta_i}$. By considering the low-load system or high-performance computing system, i.e., $n_i \cdot C_i << T$, the objective function can be relaxed to

$$U = \sum_{i=0}^{I} P_i \cdot \log \left( \frac{n_i}{C_i \cdot (n_i + 1)} \right).$$

(11)

By substituting from (11) and (7), our optimization problem turns to be in the form of Geometric Program, which can be solved efficiently using convex optimization tools [28].

2) Heuristic Approach (HA)

The main idea of this solution is to assign the upcoming new task to the instance that obtains the maximum objective, i.e., $\max \left( P_i \cdot \log(n_i/T_i) \right)$. Thus, for each new task, this approach searches for the instance that can obtain the maximum objective after assigning this task to it. We remark that this approach does not apply any relaxation for the objective function, while obtaining the best task assignment with a maximum number of iterations equals to $N$ (i.e., number of arriving tasks). The main steps in the algorithm are illustrated in Algorithm 1.

**Algorithm 1 Heuristic Approach (HA)**

0: **Input:** $(P_i, C_i, T)$, $\forall i \in \{1, \cdots, I\}$.
1: for $j = 1 : N$ do
2: for $i = 1 : I$ do
3: For each instance, calculate the objective given that it will be assigned a new task using $U(i) = P_i \cdot \log \left( \frac{n_i}{T_i} \right)$.
4: end for
5: Obtain the instance with maximum objective using $i^* = i (U)$.
6: Assign the new task to the instance with maximum objective, i.e., $n(i^*) = n(i^*) + 1$.
7: end for
8: return $n_i^* = 0$

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
V. EXPERIMENT AND ANALYSIS

This section presents the experimental setup, which includes the implementation of EPC entities adopted for this work, as well as the specifications of the VMs and configurations used for the implementation on AWS platform. The section is concluded with a discussion on the results obtained for the two implementations and the AWS task assignment optimization.

A. ENVIRONMENT SETUP

As shown in Figure 1, we implemented the OpenAirInterface (OAI) project of the LTE EPC and the RAN on AWS in order to evaluate the two proposed auto-scaling models. With OAI, containerized and non-containerized implementations of the LTE EPC entities are available along with a complete, simulated UE and eNodeB control and data plane functionalities. It should be noted that each implementation was executed within its own VPC.

Four metrics are considered to assess the scaling capability of the two auto-scaling models as follows: number of successfully attached UEs (a successful UE attach and connection process), latency (in this context, latency is defined as the duration of the attach process), and CPU and RAM consumption for containerized and non-containerized EPC entities. For the container-based implementation and the VM-based implementation, CPU usage and RAM usage are at the ECS level and at the EC2 level respectively. In both models, the auto-scaling policy threshold is set at 90% of CPU’s service usage of the entities.

Table III shows the resources assigned to the container-based and VM-based implementations. For fair comparison, we allocate the same resource capacity in terms of storage, CPU and RAM for both models. Note that the RAN is a dedicated EC2 instance of type t2.xlarge. We test the proposed auto-scaling model with many configurations and the three relevant configurations for both container-based and VM-based implementations are shown in Table IV.

B. RESULTS AND DISCUSSION

As depicted in Fig. 2, there is little to no difference in the number of registrations per second for both container-based and VM-based implementations under the three configuration settings. One noticeable trend is the increase in the number of registrations per second completed, as the number of MME entities increases in both implementations. For example, configuration 3 achieves the highest number of successful registrations per second because three MME entities were tasked to handle the attach requests of the UEs.

Fig. 3 depicts the RAM utilization metrics for 118 UEs using the container-based and VM-based implementations of the EPC entities. Under both implementations, maximum RAM utilization among the EPC entities does not exceed 670MB for all configurations. This means that as the workload varies for 118 UEs, the RAM does not saturate. This revelation has led us to assigned CPU usage as the metric of auto-scaling in both implementations.

In our evaluation of CPU utilization, we found that the MME container is the only one to provide a significant difference in CPU metrics, with 118 UEs for both container-
TABLE 3. Resource Allocation for Auto-scaling Models

| Entity | Storage | CPU | RAM GB | Storage | CPU | RAM GB |
|--------|---------|-----|--------|---------|-----|--------|
| MME    | 10 GB   | 1   | 1 GB   | 10 GB   | 1   | 1 GB   |
| HSS    | 20 GB   | 2   | 2 GB   | 20 GB   | 2   | 2 GB   |
| SPGW-C | 10 GB   | 1   | 1 GB   | 10 GB   | 1   | 1 GB   |
| SPGW-U | 10 GB   | 1   | 1 GB   | 10 GB   | 1   | 1 GB   |
| Simulated RAN | 30 GB | 4 | 16 GB | 30 GB | 4 | 16 GB |

TABLE 4. Configurations for the Auto-scaling Model for both container and VM implementations

| Entity | Config 1 (C1) | Config 2 (C2) | Config 3 (C3) |
|--------|---------------|---------------|---------------|
| MME    | 1             | 2             | 3             |
| HSS    | 1             | 1             | 3             |
| SPGW-C | 1             | 2             | 3             |
| SPGW-U | 1             | 2             | 2             |

FIGURE 5. Latency under container-based and VM-based implementations with varying number of UEs

Based and VM-based implementations. Fig. 4 shows the CPU utilization results for both implementations of MME entity. It is clear that container-based implementation has a better CPU utilization compared to the VM-based implementation in all configurations. With the number of UEs exceeding 64 under configuration 1, the auto-scaling threshold (90% percent CPU utilization of MME entity) is violated resulting in a horizontal scaling of the MME entity to handle the additional UEs. Configuration 2 (under VM-based implementation), however, was successful in serving up to 96 UEs without violating the auto-scaling policy threshold. For the container-based implementation, configuration 1 was able to handle 64 UEs without the need to scale the MME entity, while configuration 2 can handle up to 112 UEs without violating the same threshold. When 3 MME entities were deployed for both implementations, the auto-scaling policy threshold was not violated.

Fig. 5 shows the results of latency for both the container- and VM-based implementations. The result agrees with the result obtained in Fig. 2 for the registrations per second. However, it can be noticed when the number of registered UEs increases, the auto-scaling functionality adds more EPC entities (mainly the MME entity) to cope with the increment in attach requests, which makes the ELB to distribute the workload to more entities. This eventually reflects in the added latency. However, the container-based implementation still outperforms the VM-based implementation, achieving a lower latency in all configurations.

After that, we carry out a performance analysis of the proposed solutions for task assignment, namely ROS and HA, with the well-known Round Robin (RR) algorithm for task scheduling. Herein, we consider a varying number of arriving tasks within a period $T = 30$ seconds are assigned to four instances. First, to ensure that the proposed solutions can converge to the optimal solution, we consider in Fig. 6 that all instances have the same computational capabilities, i.e., it is assumed that $P_i$’s and $C_i$’s equal 1 for all instances. In this case, the RR algorithm is guaranteed to converge to the optimal solution. Then, by comparing our solutions with the RR algorithm, it can be seen in Fig. 6 that the proposed ROS and HA solutions have converged to the same solution (in terms of the throughput and number of assigned tasks for each instance) as RR algorithm, which proves the optimality of the proposed solutions.

In Fig. 7, we consider the general case where the available instances have different computational capabilities (the following parameters are assumed for the four instances: $P_1 = 0.2$, $C_1 = 1$; $P_2 = 0.3$, $C_2 = 1$; $P_3 = 0.8$, $C_3 = 0.5$; $P_4 = 1$, $C_4 = 0.2$). This figure depicts that the proposed ROS and HA solutions have the same performance and both of them outperform the RR algorithm. Indeed, with an increasing number of arriving tasks, ROS and HA obtain a much higher throughput compared to RR. We remark that the RR algorithm increases the throughput until a certain limit, then by increasing the arriving tasks, the system throughput starts to decay due to the equal assignment strategy that results in overwhelming the system by significantly increasing the sojourn time for low-computational capability instances. Unlike the proposed ROS and HA that optimally distribute the arriving tasks on the available instances based on their computational capability and sojourn time.
VI. CONCLUSION AND FUTURE RESEARCH

This paper implements two predictive auto-scalers for container-based and VM-based EPC entities on the AWS cloud environment, along with two task assignment schemes that distribute the incoming requests among diverse instances efficiently and fairly. AWS’s auto-scaling and load balancing capabilities are used in our implementation to determine when EPC entities need to be scaled to handle increased workloads. Performance evaluations reveal the superiority of container-based model over VM-based model in regard to resource utilization and the MME entity is the most critical one that needs to be scaled as the workload increases in both container-based and VM-based implementations. Furthermore, we have demonstrated how the proposed task assignment schemes could enhance the system throughput compared to RR task assignment algorithms. The proposed framework could be improved in a variety of ways, thus the possible future directions are as follows:

- **Security and Privacy:** Cloud service providers put forward a security mechanism that logically isolates traffic from different end users (for example, a UE trying to access the core network). However, such mechanisms have many flaws that can be exploited for attacks such as loss of subscribers records, compromised paging notification messages and so on [29]. Similarly, many privacy-related threats to the core network have been studied and feasible solutions were proposed to tackle them. However, the solutions are at the core network level and do not consider the case when the threats occur as the user’s traffic transverse through the internet. Therefore, a security framework that can cryptographically secure the core network’s traffic (i.e., provide an end-to-end protection of users’ traffic) is required for a secure deployment of core network entities on a public cloud environment. Software Defined Perimeter (SDP) [30] is an example of such a security framework that can be deployed alongside the core network entities to provide a zero-trust environment for safe and secure...
communications among the entities.

- **Scalability and Elasticity:** The proposed horizontal scaling mechanism scales in/out the EPC entities according to the workload variations. An interesting research investigation is to explore the elasticity of cloud resources and design an elastic auto-scaling framework which will not only scale horizontally, but also scale vertically whenever the required workload does not necessitate horizontal scaling of the EPC entity involved. Indeed, this will optimize the cloud resources more efficiently.

- **Global deployment and collaboration:** For a successful deployment, the introduction of a reference architecture to facilitate collaboration across different MEC service providers is paramount. A blockchain-based collaboration across operators has been introduced in [31] for roaming services. Following the proposed deployment auto-scaling framework for the core entities, the combination with blockchain-based agreement is feasible.

**ACKNOWLEDGMENTS**

This work was made possible by NPRP grant # NPRP12S-0305-190231 from the Qatar National Research Fund (a member of Qatar Foundation). The findings achieved herein are solely the responsibility of the authors.

**REFERENCES**

[1] Cisco Annual Internet Report (2018–2023) White Paper. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] M. Kumar, S. Sharma, A. Goel, and S. Singh, “A comprehensive survey for scheduling techniques in cloud computing,” Journal of Network and Computer Applications, vol. 143, pp. 1 – 33, 2019.

[3] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, “A comprehensive survey of network function virtualization,” Computer Networks, vol. 133, pp. 212 – 262, 2018.

[4] S. M. Sayed, S. E. Shaaban, and E. S. - A. - M. El-Badawy, “Lie network virtualization feasibility study,” in 2014 International Conference on Computer and Communication Engineering, 2014, pp. 158–161.

[5] Y. Takano, A. Khan, M. Tamura, S. Ishiwashi, and T. Shimizu, “Virtualization-based scaling methods for stateful cellular network nodes using elastic core architecture,” in 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, 2014, pp. 204–209.

[6] J. Prados-Garzon, J. J. Ramos-Munoz, P. Ameigeiras, P. Andres-Maldonado, and J. M. Lopez-Soler, “Modeling and dimensioning of a virtualized mme for 5g mobile networks,” IEEE Transactions on Vehicular Technology, vol. 66, no. 5, pp. 4383–4395, 2017.

[7] G. Premaksan, K. Ahokas, and S. Luukkainen, “Design and implementation of a distributed mobility management entity on openstack,” in 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), 2015, pp. 1 – 6.

[8] P. Satpathy, J. Dave, P. Naik, and M. Vutukuru, “Performance comparison of state synchronization techniques in a distributed lte epc,” in 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017, pp. 1–7.

[9] A. Banerjee, R. Mahindra, K. Sundaresan, S. Kasera, K. der Merwe, and S. Rangarajan, “Scaling the lte control-plane for future mobile access,” in 2014 IEEE International Conference on Cloud Computing Technology and Science, 2014, pp. 204–209.

[10] V. -G. Nguyen, K. -J. Grimmemo, J. Taheri, J. Forsman, T. Le Duc, and A. Brunström, “On auto-scaling and load balancing for user-plane functions in a softwarized 5g core,” 2021.

[11] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, “A survey on virtual machine migration: Challenges, techniques, and open issues,” IEEE Communications Surveys Tutorials, vol. 20, no. 2, pp. 1206–1243, 2018.

[12] F. Malandrino, C.-F. Chiasserini, G. Einziger, and G. Scalosub, “Reducing service deployment cost through vnf sharing,” IEEE/ACM Transactions on Networking, vol. 27, no. 6, pp. 2363–2376, 2019.

[13] A. A. Refaey, K. Hammad, S. Magierowski, and E. Hossain, “A blockchain policy and charging control framework for roaming in cellular networks,” IEEE Network, vol. 34, no. 3, pp. 170–177, 2020.
YAHUZA BELLO received his Bachelor of Science degree in Electronics and Communications Engineering from Arab Academy for Science, Technology and Maritime Transport (AASTMT) in 2014 and his Master of Science degree from Manhattan College, Riverdale, New York, USA in 2021. His research interests include software defined networks, network function virtualization, cloud computing, Network security.

Dr. Alaa Abdellatif has authored or co-authored over 35 refereed journal, conference papers and conference papers in reputable international journals and conferences. He has served as a technical reviewer for many international journals and magazines. His research interests include Edge computing, blockchain, machine learning and resources optimization for wireless heterogeneous networks, smart-health, IoT applications, and vehicular networks. He was the recipient of the Graduate Student Research Award (GSRA) from Qatar National Research Fund, and the Best Paper Award from Wireless Telecommunications Symposium 2018, in USA, in addition to the Quality Award from Politecnico di Torino in 2018.

MHD SARIA ALLAHHAM received the Bachelor of Science degree in Computer Engineering with honors from Qatar University, Doha, Qatar, in 2020. He is currently working as a Research Assistant with the Department of Computer Science and Engineering, Qatar University. His research interests include deep learning, reinforcement learning, and edge computing.

DR. AHMED REFAEY HUSSEIN (S’03-M’07-SM’15) is an Assistant Professor at Manhattan College as well as an adjunct research professor at Western University. Previously, Dr. Hussein’s positions included: a Sr. Embedded Systems Architect, R&D group, Mircom Technologies Ltd from 2013-2016; and as a Postdoctoral Fellow at ECE department, Western University from 2012-2013; and Professional Researcher at the LRTS lab, Laval University in the field of wireless communications hardware implementations from 2007-2011. Prior to joining Laval University, Dr. Hussein was a System/Core Network Engineer leading a team of junior engineers and technicians in the telecom field in the three prominent companies of Fujitsu, Vodafone, and Alcatel-Lucent. Dr. Hussein received his B.Sc. and M.Sc. degrees from Alexandria University, Egypt in 2003 and 2005, respectively; and Ph.D. degree from Laval University, Quebec, Canada in 2011. Dr. Hussein is the author and Co-author of more than 45 technical papers, 1 patent granted, and 3 patent applications addressing his research activities.

DR. AIMAN ERBAD received the Master of Computer Science degree in embedded systems and robotics from the University of Essex, Colchester, U.K., in 2005, and the Ph.D. degree in computer science from the University of British Columbia, Vancouver, BC, Canada, in 2012. He is an Associate Professor with the College of Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar. His research interests span cloud computing, edge computing, IoT, private and secure networks, and multimedia systems. Dr. Erbad received the Platinum Award from H. H. The Emir Sheikh Tamim bin Hamad Al Thani at the Education Excellence Day 2013 (Ph.D. category). He received the 2020 Best Research Paper Award from Computer Communications, the IWCMC 2019 Best Paper Award, and the IEEE CCWC2017 Best Paper Award. He is an Editor for KSII Transactions on Internet and Information Systems and a Guest Editor for IEEE NETWORKS.

DR. AMR MOHAMED (Senior Member, IEEE) received his M.S. and Ph.D. degrees in electrical and computer engineering from the University of British Columbia, Vancouver, Canada, in 2001 and 2006, respectively. He worked as an advisory IT specialist at the IBM Innovation Centre in Vancouver from 1998 to 2007, taking a leadership role in systems development for vertical industries. He is currently a professor in the College of Engineering at Qatar University and the Director of the Cisco Regional Academy. He has over 25 years of experience in wireless networking research and industrial systems development. He holds three awards from IBM Canada for his achievements and leadership, and four best paper awards from IEEE conferences. His research interests include wireless networking and edge computing for IoT applications. He has authored or co-authored over 200 refereed journal and conference papers, textbooks, and book chapters in reputable international journals, and conferences. He is serving as a technical editor for two international journals and has served as a technical program committee (TPC) co-chair for many IEEE conferences and workshops.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3126048, IEEE Access
DR. MOHSEN GUIZANI (Fellow, IEEE) received the B.S. (with Distinction) and M.S. degrees in electrical engineering, the M.S. and Ph.D. degrees in computer engineering from Syracuse University, Syracuse, NY, USA, in 1984, 1986, 1987, and 1990, respectively. He is currently a Professor with the Computer Science and Engineering Department, Qatar University, Doha, Qatar. Previously, he served in different academic and administrative positions with the University of Idaho, Moscow, ID; Western Michigan University, Kalamazoo, MI, USA; the University of West Florida, Pensacola, FL, USA; the University of Missouri–Kansas City, Kansas City, MO, USA; the University of Colorado–Boulder, Boulder, CO, USA; and Syracuse University, Syracuse, NY, USA. He has authored nine books and more than 500 publications in refereed journals and conferences. His research interests include wireless communications and mobile computing, computer networks, mobile cloud computing, security, and smart grid. Prof. Guizani received three teaching awards and four research awards. He also received the 2017 IEEE Communications Society WTC Recognition Award as well as the 2018 Ad Hoc Technical Committee Recognition Award for his contribution to outstanding research in wireless communications and Ad Hoc Sensor networks. Prof. Guizani is currently the Editor-in-Chief of the IEEE Network Magazine. He serves on the editorial boards of several international technical journals and is the Founder and Editor-in-Chief of Wireless Communications and Mobile Computing (Wiley). He guest edited a number of special issues in IEEE journals and magazines. He also served as a member, chair, and general chair of a number of international conferences. He was the Chair of the IEEE Communications Society Wireless Technical Committee and the Chair of the TAOS Technical Committee. He served as the IEEE Computer Society Distinguished Speaker and is currently the IEEE ComSoc Distinguished Lecturer. He is a Senior Member of ACM.

***