Super-Cloudlet: Rethinking Edge Computing in the Era of Open Optical Networks

Behzad Mirkhanzadeh*, Tianliang Zhang*, Miguel Razo-Razo*, Marco Tacca*, and Andrea Fumagalli*
*Open Networking Advanced Research (OpNeAR) Lab, UT Dallas, TX, USA
{behzad, tianliang.zhang, mrazo, mtacca, andreaf}@utdallas.edu

***Abstract*—Edge computing is an attractive architecture to efficiently provide compute resources to many applications that demand specific QoS requirements. The edge compute resources are in close geographical proximity to where the applications' data originate from and/or are being supplied to, thus avoiding unnecessary back and forth data transmission with a data center far away. This paper describes a federated edge computing system in which compute resources at multiple edge sites are dynamically aggregated together to form distributed super-cloudlets and best respond to varying application-driven loads. In its simplest form a super-cloudlet consists of compute resources available at two edge computing sites or cloudlets that are (temporarily) interconnected by dedicated optical circuits deployed to enable low-latency and high-rate data exchanges. A super-cloudlet architecture is experimentally demonstrated over the largest public OpenROADM optical network testbed up to date consisting of commercial equipment from six suppliers. The software defined networking (SDN) PROnet Orchestrator is upgraded to both concurrently manage the resources offered by the optical network equipment, compute nodes, and associated Ethernet switches and achieve three key functionalities of the proposed super-cloudlet architecture, i.e., service placement, auto-scaling, and offloading.**

I. Introduction

The continuous growth of Internet of Things (IoT), connected devices and vehicles, cloud robotics, and 5G wireless communications has led to a pervasive computing infrastructure with a variety of peripheral devices that act as a set of heterogeneous data sources and offer a wide range of services [1] [2]. Cloud computing — a term generally used to describe data centers — is a key enabler for such services as it provides a straightforward way to enhance the computational capabilities of said peripheral devices by offering a centralized data center that is rich of compute resources. As cloud-based services evolve, Quality of Service (QoS) requirements, e.g., network latency and data rate, needed by such services tend to become more stringent. For example, real-time cloud services in industry 4.0 [3], patient monitoring, and self-driving vehicles often require low-latency and in some cases high-data rate network solutions. The traditional cloud computing infrastructures, i.e., a centralized (and usually remote) data center, cannot always fulfill these QoS requirements due to the inherent latency of the wide area network (WAN) connection needed to access the data center [4]. Furthermore, hosting the computing resources in smaller sites that are in close proximity to the user-data has the additional advantage of reducing the data traffic in WANs and the potential to lower the service cost charged to the end-users [1] [5].

These considerations have led to the formulation of decentralized computing systems — like Mobile Cloud Computing (MCC) [6], Cloudlet [7], fog computing [8], and edge computing [9] — which are designed to best address both QoS requirements and cost of computational-intensive services at the edge. Applications, often referred to as tenants, can request resources to be reserved at nearby edge computing sites in addition to what centralized cloud computing already offers.

While edge computing has the potential to meet the tenants' tightest QoS constraints, the resulting distributed architecture presents new challenges. Edge sites (also referred to as cloudlets) host considerably fewer computational resources compared to conventional cloud data centers [1]. Even when carefully planned, cloudlet compute resources may not always suffice to meet an unexpected spike of application-driven load. Offloading part of the load spike to a centralized data center may not always be viable as it could violate the tight QoS requirements of some applications. An alternative solution is to offload part of the load spike to another cloudlet. For example, a nearby cloudlet could share some of its unused computational resources with the cloudlet experiencing the load spike in a seamless manner, provided that the two cloudlets are interconnected by means of a low-latency and high-rate data network connection.

This paper describes and experimentally showcases a federated edge computing architecture in which compute resources at multiple physical cloudlets are dynamically aggregated together to form *distributed super-cloudlets*. The objective of creating super-cloudlets dynamically is to best respond to time-variant loads offered by edge-supported applications without having to resort to permanently over-provisioned cloudlets. In its simplest definition a super-cloudlet comprises compute resources that are available at two physically distinct cloudlets[1]. These resources are temporarily and seamlessly combined together to form a larger and distributed super-cluster of compute nodes. The newly formed distributed super-cluster is more likely able to successfully cope with peaks of offered load while still fulfilling the QoS requirements that the edge-supported applications demand. To this end the cloudlets that are chosen to form the super-cloudlet must be interconnected with a low-latency and high-data rate network connection, which may be either statically or dynamically

[1]Extension of the super-cloudlet concept to three of more cloudlets is outside the scope of this study.

provisioned in the metro area network (MAN). Optical circuits are best suited for this type of connectivity [10].

For example, this low-latency and high-data rate network connectivity is achievable through dense wavelength division multiplexing (DWDM) equipment that is compliant with the public OpenROADM Multi Source Agreement (MSA) standard [11].

Such an open solution offers cloudlet operators the ability to independently choose their own preferred supplier, yet permitting cloudlets of different operators to be directly connected by optical circuits when required.

To showcase the proposed concept, an OpenROADM testbed is programmed to implement super-cloudlets that support a machine learning training tenant application whose performance is sensitive to architectural constraints.

II. State of the Art

The concept of forming federations of multiple cloudlets has been theoretically investigated in a number of papers [12]–[14]. Ref. [12] focuses on optimal planning of the optical access network that interconnects the attached cloudlets. A non-linear programming model is proposed to identify optimal placement locations of cloudlet servers subjected to capacity and latency constraints in urban, suburban, and rural scenarios. Ref. [13] describes an economic and non-cooperative continuous-kernel game theoretic model to analyze computation offloading strategies between cloudlets of competing service providers, interconnected by a TDM optical access networks.

Ref. [14] describes a small cell base station (SBS) coalition formation algorithm that is based on the coalitional game theory to cope with various challenges in small-cell-based edge systems, including the co-provisioning of radio access and computing services, cooperation incentives, and potential security risks. This simulation-based study shows that significant edge computing performance improvement is achievable by allowing cooperation among SBSs. Other studies in the literature look into load balancing and auto-scaling schemes in compute infrastructures [15]–[17] without paying particular attention to the required optical network infrastructures.

Published studies on federated cloudlets are mostly theoretical and focus on optical access networks. In contrast, the study described in this paper focuses on the first experimental work on federated cloudlets interconnected by a programmable (SDN) DWDM optical network testbed.

III. System Description

In this section, we describe a possible implementation of the super-cloudlet architecture along with a related testbed that is used to conduct a number of experiments to validate the key required functionalities. The described implementation is a compromise between proving these functionalities and making use of a number of open software packages that are available at the time of this publication. Over time, this specific implementation may be further improved based on future upgrades of these software packages.

As previously mentioned, besides offering a decentralized and close-to-the-application service, edge computing must also represent a cost effective alternative to centralized cloud computing. More specifically, edge computing must address:

- High quality of service: the solution must support service to applications with low latency requirement by assigning compute resources that are geographically close (in a cloudlet) to the applications [5].
- Low WAN cost: the solution must reduces congestion in the WAN by minimizing the data exchange required between decentralized cloudlets and centralized cloud computing sites — e.g., offloading services to a centralized cloud should be avoided whenever possible [1].
- Real time scaling: similar to conventional cloud computing a cloudlet also needs to provide elasticity to scale up and down operation based on the number of active users and applications running [1].
- Service diversity: cloudlets must be able to offer a set of diverse multi-tenant compute services in the form of virtual machine, container, and bare metal [5].

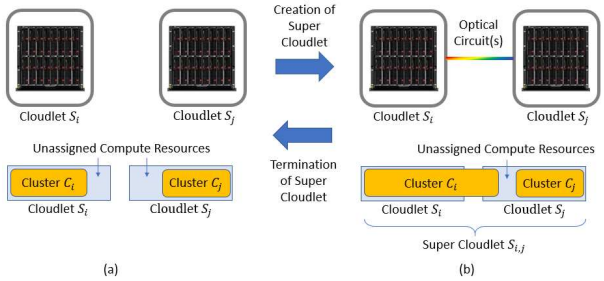
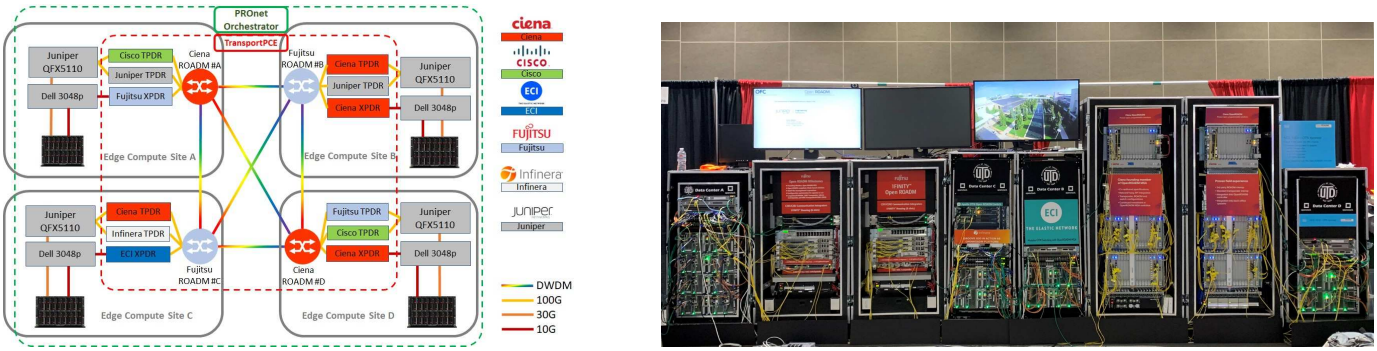


Fig. 1. (a) Two cloudlets operating independently, each hosting a single cluster of workers. (b) A super-cloudlet consisting of two cloudlets connected by low-latency and high-data rate optical circuits. Worker cluster C_i makes use of compute resources that are provided by cloudlet S_j to cope with a sudden surge of its applications' load.

Consisting of two neighboring cloudlets interconnected by low-latency and high-dat rate optical circuits the super-cloudlet is designed and implemented to concurrently fulfill all of the above edge computing requirements. Fig. 1(a) depicts two cloudlets (S_i and S_j) that work independently, each hosting one cluster of workers (C_i and C_j, respectively). Fig. 1(b) shows how the creation of a super-cloudlet ($S_{i,j}$) enables cluster C_i to scale up above its hosting cloudlet (S_i) capacity by resorting to the additional compute resources that are available at the nearby cloudlet (S_j). This procedure enables real-time scale up of cloudlet compute resources while keeping them close to the applications, does not increase traffic in the WAN, and supports service diversity as described next.

Our testbed implements a federated edge computing system with four cloudlets interconnected by a packet SDN over a full mesh OpenROADM [11] transport network. The topology schematic is shown in Fig. 2a and the actual equipment is shown in Fig. 2b. A total of forty re-purposed Stampede compute nodes [18] are divided among the four cloudlets and controlled through OpenStack [19]. Four OpenFlow-enabled



(a) Testbed block diagram.

(b) Photo of testbed equipment as displayed at OFC'20.

Fig. 2. Live demonstration of the OpenROADM testbed with four nodes.

Juniper QFX[2] and four Dell[3] Ethernet switches provide connectivity between the compute nodes and the optical transponders and switchponders. The switches are controlled by an OpenDaylight (ODL) controller. The optical transport network consists of four ROADM nodes, one for each cloudlet [20]. Length of the fibers connecting the ROADMs can be manually modified to investigate the effect of signal propagation on the system performance. Data packets from the Ethernet switches are transmitted over the optical transport network in the form of client signals using eight 100Gbps transponder blades (for a total of 20 wavelengths), and five switchponders (aggregate line rate of 100Gbps and client rates of 1Gbps and 10Gbps). Additional details about the three domains in the testbed are provided next.

A. Compute Domain

In its most general definition edge computing should offer services on various platforms such as virtual machine, containers, and bare metal. Virtual machines and bare metal both provide more mature and isolated compute environments for cloud-driven applications. Containers are increasingly being considered for computationally-intensive and distributed tasks because they offer more agile software platforms to developers like DevOps and microservices [21]. Due to this growing interest our study focuses on containerized applications.

In order to run containerized applications in a federated computing edge system a container orchestration engine (COE) [22] is required, which enlists compute resources to form a *cluster of workers* and assigns users' tasks (in containers) to the available workers in the cluster. The COE usually manages key functionalities like task placement, auto-scaling, and load balancing for the cluster and keeps administration of the system simple. Examples of COE include Kubernetes [23], Docker Swarm [24] and Apache Meso [25], which all support these key functionalities [22]. A COE cluster typically consists of at least one *master node* and one *worker*. The master node performs all the administrative and controlling functions, while the workers host tasks running as containers. COEs do not typically have awareness of the cloudlets locations, network topology and equipment configuration, and do not natively support super-cloudlets. These functionalities — missing in existing COEs — must be implemented at a higher level of control, like a comprehensive platform for orchestration, management, and automation of network and edge computing services. This orchestration platform — e.g., ONAP [26] and the PROnet Orchestrator [27] — has the necessary visibility across both network and compute domains and is therefore most suitable for implementing the key functionalities that are required in the super-cloudlet architecture, like setting up and taking down super-cloudlets.

In our testbed, OpenStack [4] is chosen to provide infrastructure as a service platform (IaaS), which includes enlisting of the compute nodes and maintenance of networking and multi-tenancy across cloudlets. OpenStack firstly supports multiple COEs to host containerized applications through the Magnum API service. Secondly, OpenStack offers both virtual machine and bare-metal service in the cloudlets, thus achieving the service diversity discussed in III. Each COE is implemented using Kubernetes[5] . For ease of description only one cluster of workers (C_i) is created in each cloudlet (S_i), where $i = 1, 2, 3, 4$. Two options exist in OpenStack two host Kubernetes workers: 1) bare-metal using Ironic and 2) virtual machine using KVM. These two options have their pros and cons. While the bare metal option provides hosting with reduced overhead and improved performance, the virtual machine option offers more flexibility for workers to be live migrated for load balancing and offloading purposes. Being live migration a required key functionality to support super-cloudlets, the latter option is chosen in this study. Fig. 3 depicts an individual cloudlet and how the Kubernetes cluster is hosted inside OpenStack. Network connectivity is described next.

B. Network Domain

A typical edge computing infrastructure requires two separate main networks.

- The *management network* is required for administrative services such as load balancing of compute nodes and exchange of control messages for various services and API access.

[2]Juniper QFX5100 Series 10/25/40/100GbE Switches.
[3]Dell PowerConnect 3048P Switches.
[4]OpenStack Stein release.
[5]Kubernetes version 1.17

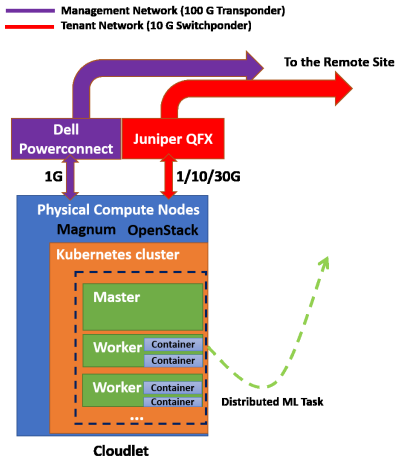


Fig. 3. Kubernetes cluster in federated cloudlet.

- The *tenant network* is commonly known as production network for West/East traffic in cloud and edge, and usually provides dedicated communication for computing instances that belong to a specific tenant. Tenant networks are usually overlay networks such as generic routing encapsulation (GRE) [28] and VxLAN [29].

It is worth mentioning that a third network type referred to as *external network* carries North/South traffic, i.e., the traffic between the compute resources and the peripheral devices that belong to the tenant. The tenant devices are connected to one or more external networks and the compute instances must be able to connect to these networks so tenants can have data exchanges between their devices and assigned compute instances.

When a super-cloudlet is created both management and tenant networks must be upgraded to account for the distributed nature of the compute resources hosted by the two cloudlets that are being aggregated. For best performance, cost effectiveness, and solution modularity dedicated optical circuits are (dynamically and temporarily) provisioned (through wavelength and/or time division multiplexing) between these cloudlets, as shown in Fig. 1(b). Optical circuits provide the low-latency and high-data rate connectivity between the two cloudlets which is advantageous for two reasons: first, a high-performing tenant network between the cloudlets maintains the high quality of service that is expected from edge computing even when compute resources are allocated in the nearby cloudlet; second, a high-performing administrative network between the cloudlets facilitates and expedites critical procedures like task offloading and load balancing between the cloudlets.

Both packet and optical domains in the testbed are described next.

1) Packet Domain: OpenFlow-enabled packet-switching fabrics are used to commission both management and tenant networks in every cloudlet. The Juniper switches are configured to support the permanent management network, with the temporary upgrades provided by the 100G transponders when operating a super-cloudlet. The Dell switch are configured to support the permanent tenant network, with the temporary upgrades provided by the switchponders 10G client signals when operating a super-cloudlet.

The higher bandwidth of the Juniper switch enables the management network to expedite worker offload procedures, in response to disaster occurrence or real-time changes in the super-cloudlet design. Concerning tenant networks, isolation (slicing) of network resources is achieved through VXLAN which offers ease of deployment and scalability.

The VXLAN header contains a 24-bit network identifier called VNI, which enables a large number of isolated tenant networks to be established. In addition, since VXLAN is an overlay network based on UDP, tenant networks can be created on top of any layer (2 or 3) network infrastructure. Fig. 4 shows an example of VXLAN overlay network for two distinct tenants created on top of a physical network.

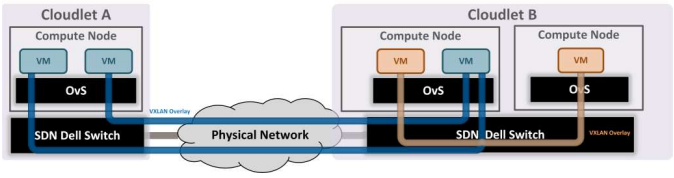


Fig. 4. Two VXLAN overlay tenant networks.

2) Optical Domain: While optical circuits between cloudlets may be permanently provisioned in special situations, a more general and flexible architecture makes use of on-demand and dynamically provisioned optical circuits. In the latter case optical circuits are provisioned only when the super-cloudlet is needed, thus yielding a significant network cost reduction. In addition, each cloudlet can be equipped with one or just a few optical transponders, and yet that cloudlet could be paired with any of the nearby cloudlets that are equipped with *compatible transponders*. Till recently this requirement of having compatible transponders deployed across heterogeneous cloudlets would have required a group of cloudlet providers to agree to use the same optical equipment supplier. However, thanks to the recent progress made by the OpenROADM MSA initiative it is now possible to achieve full interoperability between equipment from multiple suppliers, along with full SDN programmability of said equipment as discussed later in the paper. With this significant advantage of being able to create optical circuits on demand and between many possible cloudlet pairs, the operators have an unprecedented number of options to scale up their cloudlet capacity while taking into account geographical proximity of cloudlets, compute resource type and availability, and existing commercial agreements with other operators for the mutual exchange of resource usage.

In our testbed, ROADM nodes, transponders, and switchponders from six suppliers are interconnected as shown in Fig. 2a.

The resulting multi-supplier optical transport network is

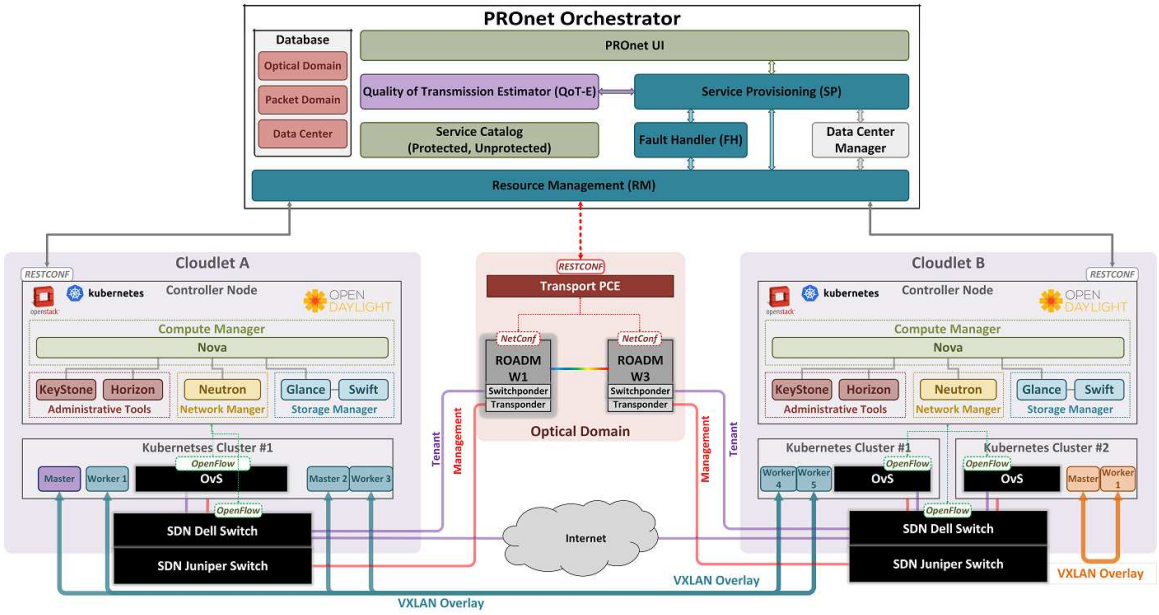


Fig. 5. PROnet Orchestrator in federated edge computing.

controlled by the open source TransportPCE plugin[6] [30], which makes use of a common set of NETCONF/YANG APIs published by the OpenROADM MSA[7]. [11]. The OpenROADM MSA specifications ensure interoperability of ROADM, transponders, OTN switches, and pluggable optics. This OpenROADM MSA compliance is a very critical element to achieve a multi-operator federation of cloudlets in which each operator is free to select its own equipment supplier, and yet optical end-to-end dynamic connectivity between cloudlets remains feasible.

As already mentioned, while operating a super-cloudlet transponders with 100G client ports are used to provide temporary extension of the management network and switchponders with 1G and 10G client ports are used to provide temporary extension of the tenant networks. This is an arbitrary choice made in our study, driven by two observations: a high data rate (e.g., 100G) management network expedites migration of workers between cloudlets, and switchponders make use of time division multiplexing (OTN [20]) to create multiple isolated client circuits with relatively low rates (e.g., 1G and 10G) over a single (e.g., 100G) optical circuit, which enable guaranteed bandwidth separation among multiple tenant networks. With that noted, thanks to the software programmability of OpenROADM networks the operator can choose to assign other transport circuit rates to interconnect cloudlets. For example 400G transponders compliant with OpenROADM are expected to become available soon, and demanding tenant applications may be assigned data rates in excess of 10G when necessary.

[6]TransportPCE plugin version 2.0.0
[7]OpenROADM MSA version 2.2.1

C. Orchestration architecture

The resources in the three domains (compute, electronic packet switching, and optical circuit switching) are managed and orchestrated by a single software module — whose block diagram is shown in Fig. 5 — referred to as the PROnet (SDN) Orchestrator [27]. Interfaced with four controllers — the TransportPCE and OpenFlow ODL controllers for managing network resources, and the OpenStack and Kubernetes controllers for managing compute resources — the PROnet Orchestrator is configured to execute three key functionalities of the super-cloudlet architecture, i.e., service placement, auto-scaling, and service offloading. In order to do that, the PROnet Orchestrator performs resource discovery and imports resource states from the four controllers by means of the RESTCONF protocol. Based on its internal combined abstraction of the three domains and their resources — which also keeps track of every super-cloudlet that is created and its current state — the PROnet Orchestrator is then able to implement the schedulers that activate the three key functionalities and, when necessary, trigger the creation or removal of super-cloudlets.

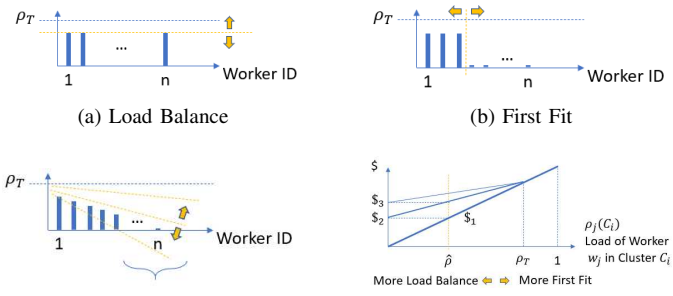
The creation of a specific super-cloudlet should be driven by algorithms that make use of equations which are based on the following parameters and variables[8]. Let S_i represent a cloudlet. Let C_i be the COE cluster hosted by cloudlet S_i. The following notation is defined:

- $\sigma(S_i)$=$\{n(S_i), c(S_i), m(S_i), s(S_i)\}$: Resource provisioned at cloudlet S_i defined as the number of compute nodes, vCPUs per node, memory per node, and storage per node,

[8]The set defined in this study only provides a simple example to help illustrate the concept. This minimal set can be expanded to account for many more factors that play key roles in practical deployments.

- $\gamma(C_i)$=$\{c(C_i), m(C_i), s(C_i)\}$: Worker flavor for cluster C_i defining the number of vCPUs, memory, and storage requirements per worker,
- $w(C_i)$: Number of workers reserved for C_i,
- $c_j^u(C_i) \leq c_j(C_i)$: Number of actively used vCPUs in worker w_j in C_i,
- $m_j^u(C_i) \leq m_j(C_i)$: Memory actively used in worker w_j in C_i,
- $s_j^u(C_i) \leq s_j(C_i)$: Storage actively used in worker w_j in C_i,
- $\rho_j(C_i)$=$max\left(\frac{c_j^u(C_i)}{c_j(C_i)}, \frac{m_j^u(C_i)}{m_j(C_i)}, \frac{s_j^u(C_i)}{s_j(C_i)}\right)$: Utilization factor of worker w_j in C_i,
- $\rho(C_i)$=$\sum_{j=1}^{w(C_i)} \rho_j(C_i)/w(C_i)$: Average utilization factor of all workers in C_i, assuming all workers have identical flavors,
- $\sigma(S_{i,j})$=$\{\sigma(S_i), \sigma(S_j)\}$: Resource provisioned in super-cloudlet $S_{i,j}$, defined as the resources that have been built in the two cloudlets comprising the super-cloudlet, i.e., S_i and S_j,
- ω_l=$\{c(\omega_l), m(\omega_l), s(\omega_l)\}$: Flavor for container of type l, defining number of vCPUs, memory size, and storage size requirements, $i = 1, 2, 3, 4, \ldots$,
- Ω_m=$\{$k,$\omega_l\}$: Task request type, consisting of k containers of type m.

Building on these parameters and variables scheduling algorithms can be designed to assist both the COE and orchestration platform to carry out service placement, auto-scaling, and service offloading. Implementation of these three key functionalities in the PROnet Orchestrator is discussed next.



(a) Load Balance (b) First Fit

(c) Hybrid (d) Proposed approach

Fig. 6. Scheduling and placement strategies.

1) Service Placement and Scheduling: Service placement is the action of assigning application tasks to run on specific workers in each cluster. This scheduling decision is typically made by the COE, which accounts for the available workers in the cluster, their flavors, current and forecast load. The cloudlet administrator may also drive the COE decision based on specific preferences and strategies, e.g., achieving balanced loading of the available workers, or turning off as many workers as possible. These are conflicting strategies that must be resolved by the cloudlet administrator. Typically, incoming task request Ω_l=$\{$k,$\omega_l\}$ is hosted in cluster C_i as long as a service placement can be found such that $\rho_j(C_i) \leq T$ for every worker w_j, $j = 1, 2, \ldots$, where T is a chosen threshold. To meet the condition imposed by T it may be necessary to place the k containers in multiple workers. When multiple solutions exist for an incoming task request that satisfy the condition imposed by T, the scheduler may rank said solutions according to the administrator's preference, e.g., load balancing or first fit. For instance, the administrator might prefer a uniform load balancing among all the workers or switch to the first fit approach to minimize the cluster size within the cloudlet and in some cases avoid creating a super-cloudlet.

To account for these two conflicting conventional strategies — i.e., uniform load balance shown in Fig. 6(a) or first fit shown in Fig. 6(b) — and possible hybrid options in between — as shown in Fig. 6(c) — a simple model is described next, which is tunable through a single parameter $\hat{\rho}$. (Note that both full load balance and first fit are special cases of the hybrid option.) This model is easy to use when running various experiments as it permits to modify the service placement strategy by varying the value of $\hat{\rho}$.

Let ρ_T be a threshold assigned to workers in cluster C_i that are expected to operate at utilization levels below said threshold, i.e., $\rho_j(C_i) = \rho_j \leq \rho_T$. Let $0 \leq \hat{\rho} \leq \rho_T$ be an adjustable parameter that is set by the cloudlet administrator to shape the load distribution across its cluster workers. Parameter $\hat{\rho}$ is used to compute the utility function $y_j(\rho_j)$ for each worker w_j in cluster C_i as illustrated in Fig. 6(d). The utility function for the first worker $y_1(\rho_1)$ grows linearly with the worker load ρ_1. The utility of the second worker $y_2(\rho_2)$ is a piecewise function with two segments as shown in Fig. 6(d). The offset and slope of the left segment is determined by the value assigned to $\hat{\rho}$ using the following equations, which for the generic worker w_j is:

$$y_j(\rho_j(C_i)) = \frac{y_{j-1}(\rho_T) - y_{j-1}(\hat{\rho})}{\rho_T} * \rho_j(C_i) + y_{j-1}(\hat{\rho}). \quad (1)$$

Using the utility functions in (1) the ordered list $L(C_i)$ is created by sorting the workers in C_i from the lowest to the highest value. The utility function of each worker w_j is computed at the load that would result from assigning the container request to that worker, referred to as ρ_j^+. The lowest cost worker in the list is chosen to host the container. Note that when $\hat{\rho}$ is set to approach 0, all workers tend to have the same utility function and the resulting average load of the workers approaches the distribution in Fig. 6(a). When $\hat{\rho}$ is set to approach ρ_T, workers are loaded in a first fit manner as shown in Fig. 6(b). Intermediate hybrid distributions can be obtained by choosing $\hat{\rho}$ to be somewhere between 0 and ρ_T, obtaining average loads like those in Fig. 6(c).

2) Auto-scaling: Auto-scaling is the action of either increasing or decreasing $w(C_i)$, which is the size of cluster C_i. Auto-scaling is required to adjust the cluster resources to best meet the applications' aggregate offered load when that varies over time. Auto-scaling within an individual cloudlet is straightforward and can be performed by the orchestrator platform based on specific conditions. A typical condition for increasing $w(C_i)$ is $\rho(C_i)$ approaching threshold T. A simple

Algorithm 1 Simple scheduler algorithm.

procedure SCHEDULETASK(Ω_{k,ω_l})
  for k in Ω **do**
    if scheduleContainer(k) is $Flase$ **then**
      remove all containers of this task
      put task request in waiting queue
      scaleup()
      return $False$
    end if
  end for
  return $True$
end procedure
procedure SCHEDULECONTAINER(k_{ω_l})
  for `each` w_j `in` C_i **do**
    $\rho_j^+(C_i) \leftarrow MAX(\frac{c_j^u(C_i)+c(\omega_l)}{c_j(C_i)}, \frac{m_j^u(C_i)+m(\omega_l)}{m_j(C_i)}, \frac{s_j^u(C_i)+s(\omega_l)}{s_j(C_i)})$
    if $\rho_j^+(C_i) <= 1$ **then**
      add w_j to list of potential candidates L_{C_i}
    end if
  end for
  if L_{C_i} is Empty **then**
    return $False$
  end if
  $w_m \leftarrow \min cost(L_{C_i})$
  assign container k_{ω_l} to worker w_m
  return $True$
end procedure
procedure SCALEUP()
  if resources exist in cloudlet S_i **then**
    create and add worker to C_i
  else
    select best auxiliary cloudlet S_j for super-cloudlet
    create lightpath betweeen S_i and S_j cloudlets
    create and add worker to C_i
  end if
  return $True$
end procedure

reactive method can be defined in a way that when an incoming task cannot be scheduled in the cluster due to the lack of resources a scale up request is triggered. A proactive method monitors the state of each cluster periodically and triggers a scale up or scale down action based on the workers' utilization factors and predefined thresholds.

Auto-scaling of a cluster (C_i) size beyond its cloudlet (S_i) capacity requires the creation of a super-cloudlet. The auxiliary cloudlet (S_j) needed for creating the super-cloudlet must be selected by the orchestrator while taking into consideration the following factors at S_j: availability of compute resources, availability of optical transponders, and transport network round trip time with respect to cloudlet S_i. Other factors that may affect the selection of the auxiliary cloudlet include past history of application load distribution, future load prediction tools and collaborative agreements between cloudlets' providers. The super-cloudlet is formally set up once cloudlet S_i and S_j are connected by a dedicated optical circuit(s), which minimizes the transport latency of both management and tenant networks connecting the workers in the two sites.

In the testbed, auto-scaling is invoked when an application task request cannot be fulfilled with the workers that are currently assigned to the cluster (C_i). A task Ω_{k,ω_l} is fulfilled when k containers of flavor ω_l can be hosted by the workers available in the cluster. Algorithm 1 shows the pseudocode of a reactive auto-scaling scheduler implemented in the PROnet Orchestrator. The scheduler first attempts to scale up cluster C_i with a new worker created in the home cloudlet (S_i). When that is not possible the PROnet Orchestrator selects a nearby cloudlet to serve as auxiliary cloudlet (S_j), establishes two optical circuits (a 100G for the management network and a 10G for the tenant network), and finally scales up C_i with a new worker created in the auxiliary cloudlet. It is worth noting that the two optical circuits of the super-cloudlet are created by the TransportPCE controller *while concurrently* the cluster is scaled up by the OpenStack controller. These two tasks can proceed concurrently, since the OpenStack compute node needs mostly internet (or a local docker registry) access to download the software packages needed to deploy the new worker and does not require dedicated optical circuits with the other cloudlet to carry out this task. A proactive auto-scaling scheduler is also implemented in the PROnet Orchestrator. In this second implementation the scheduler i) checks the state of each cluster at regular intervals of $t_{cluster}$, and ii) evaluates whether the two optical circuits in the super-cloudlet should remain in place or be removed at regular intervals of $t_{lightpath}$. Two cluster utilization thresholds $\rho(U)$ and $\rho(L)$ are used by the proactive scheduler to scale up and scale down cluster C_i, respectively. If $\rho(C_i) > \rho(U)$, then C_i is scaled up. If $\rho(C_i) < \rho(L)$ and the worker in cluster C_i with the highest identifier (i.e., most recently created) is idle, then C_i is scaled down by removing said worker. When C_i no longer makes use of workers in the auxiliary cloudlet (S_j) the super-cloudlet and its optical circuits are taken down.

To avoid that both auto-scaling and optical circuit creation/removal procedures are invoked too often, a minimum life time (MLT) is enforced on the created workers and optical circuits. Once created and assigned to its cluster a worker cannot be removed before a minimum life time $t_{workerMLT}$. Once created and assigned to a super-cloudlet an optical circuit cannot be torn down before a minimum life time $t_{lightpathMLT}$.

The total time required to complete both the set up of a new super-cloudlet and cluster auto-scaling action can be reduced when the orchestrator platform is designed to carry out these two tasks in parallel. The time required to create an optical circuit — the predominant time factor in creating the super-cloudlet — depends on the total fiber distance traveled by the signal and the specific equipment used. Typically, this time is in the 3 to 5 minute range. The time to add workers to a cluster depends on the virtualization platform and compute hardware deployed. A typical value is several minutes (see Section IV),

which is comparable to the lightpath creation time. By carry out these two tasks in parallel the orchestrator can make the auto-scaling time practically transparent to the creation (or removal) of super-cloudlets.

3) Service Offloading: Service offloading is the action of migrating tasks from their currently assigned compute resources to another host. In addition to its typical applications already described in the Introduction, this functionality is also key in the super-cloudlet architecture as illustrated by the next example.

A cluster in a super-cloudlet consisting of two cloudlets (home and auxiliary) may have workers assigned to application tasks in both cloudlets. Workers in one cloudlet may need to be live migrated to the other cloudlet. For example when it is time to take down the super-cloudlet the workers that are running in the auxiliary cloudlet must be migrated to the home cluodlet, before the optical circuits between the two cloudlets are torn down.

The PROnet Orchestrator is designed to carry out live migration of virtual machines hosting workers from one compute node to another. Fig. 7, shows the two methods for live-VM migration available in OpenStack: pre-copy and post-copy methods. With the pre-copy method the memory contents of the VM are sent to the destination VM interactively while new pages might be added to the memory of the current VM. The OpenStack pre-copy strategy offers an auto-converge feature for a successful live migration of a memory-intensive VM by slowing the instance down [31]. With the post-copy method the memory content is copied to the new VM and the current VM is paused. This feature activates the virtual machine on the destination host before all of its memory has been copied. When the virtual machine accesses a page that is missing on the destination host, the resulting page fault is resolved by copying the page from the source host [31]. The performance of the two methods when applied to a super-cloudlet is discussed in the next section.

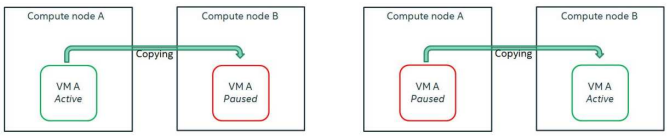


Fig. 7. Pre-copy (left) and Post-copy (right) methods for live VM migration.

IV. Experimental Results

A number of experiments are reported in this section to provide further insights into the proposed super-cloudlet architecture and its feasibility. Only three of the testbed cloudlets (A, B, and C) are used in these experiments and their configurations are reported in Table I. The experiments and their results are organized in three subsections: auto-scaling of Kubernetes cluster under varying conditions, performance of a distributed machine learning (ML) training application, and service offloading achieved by OpenStack when migrating a VM hosting containers.

A. Auto-scaling

In this series of experiments the PROnet Orchestrator instructs Kubernetes to perform scale up and scale down upgrades of a cluster hosted by a single cloudlet fifty times. Table II shows both mean and variance of the completion time of these trials distinguishing between two management network interface rates at the compute node, i.e., 1G and 30G. The auto-scale completion time is minimally affected by the management network interface used at the node. The high variance of the scale up results is mainly due to the public Internet access that is required to retrieve the required software packages and docker images for deploying the workers.

When the scale up and scale down procedures trigger a super-cloudlet set up or take down, at least one new lightpath must be created or torn down, respectively. Table III reports the average time it takes to create and tear down a lightpath in our OpenROADM network alongside the time it takes to have L3 reachability through the attached packet switches [32]. The total time required to set up the network connection needed by the super-cloudlet is 188 seconds (lightpath setupe time + L3-reachability time), which is less than the scale up time of adding one worker to the compute cluster. Since the both network set up and compute scale up procedures are executed in parallel, the total average time to create the super-cloudlet is around 420 seconds, which is the mean scale up time of the compute domain. Concerning the the scale-down procedure the 88 seconds needed to tear down a lightpath exceed the compute cluster scale down time. Thus, the total time that is required to take down the super-cloudlet is 88 seconds on average.

B. The Use Case: a Machine Learning Training Task

As previously noted, operating with a super-cloudlet has the advantage of temporarily offering extra workers to a cluster that is experiencing a load surge. However, with the cluster workers now being hosted in two cloudlets connected by an optical transport network performance degradation of the tasks running in the containers hosted by these workers may be experienced. A use case is described and used in this section to explore this important aspect. The chosen application task is the distributed training of deep neural networks. Compared to centralized training running on a single machine, distributed training achieves higher training efficiency for deep neural networks[9] by distributing the training task to run on multiple servers through either data parallelism or model parallelism [34]. More specifically, data parallelism in TensorFlow environment [35] is used in this study by running containers in either an isolated cloudlet or a super-cloudlet to train ResNet56 [36] on dataset CIFAR10 [37]. The CIFAR10 dataset contains 50,000 samples. One round of training on the whole dataset is defined as one epoch. In each epoch, the dataset is divided into multiple smaller disjoint batches that are evenly assigned to the containers hosted by the workers.

[9]Edge computing is expected to support many applications that require machine learning [33].

TABLE I
THREE CLOUDLETS USED IN THE EXPERIMENTS.

	Cloudlet A	**Cloudlet B**	**Cloudlet C**
Number of compute nodes	11	12	4
Number of CPU cores	16 (32 via Hyperthreading)	16 (32 via Hyperthreading)	16 (32 via Hyperthreading)
Memory	32GB	32GB	32GB
Storage	1TB	1TB	1TB
Compute node's management interface	1Gb/s (8 nodes) 30Gb/s (2 nodes) 10Gb/s (1 node)	1Gb/s (8 nodes) 30Gb/s (3 nodes) 10Gb/s (1 node)	30Gb/s (3 nodes) 10Gb/s (1 node)
Compute node's tenant interface	1Gb/s	1Gb/s	1Gb/s
Management switch	Juniper QFX 5110	Juniper QFX 5110	Juniper QFX 5110
Tenant switch	Dell Powerconnect 3048	Dell Powerconnect 3048	Dell Powerconnect 3048
Management optical lightpath	Transpoders (100 Gb/s)	Transpoders (100 Gb/s)	Transpoders (100 Gb/s)
Tenant optical lightpath	Switchponder (10 Gb/s)	Switchponder (10 Gb/s)	Switchponder (10 Gb/s)

TABLE II
AUTO-SCALING COMPLETION TIME AVERAGE AND VARIANCE AS A FUNCTION OF MANAGEMENT NETWORK DATA RATE.

	Mean Scale up	**Scale up Variance**	**Mean Scale Down**	**Scale Down Variance**
1G nodes	420.58s	217.22s	70.90s	13.91s
30G nodes	420.13s	184.13s	68.17s	15.90s

TABLE III
AVERAGE NETWORK CONFIGURATION TIME FOR SUPER-CLOUDLET SET UP AND TAKE DOWN.

$t_{lightpath\_creation}$	$t_{lightpath\_teardown}$	$t_{L3\_Reachability}$
178s	88s	10s

The epoch time is also divided into m consecutive steps. During a step, each container performs training using one m^{th} of its assigned batch of data. At the end of each step containers exchange their respective output data to aggregate gradients in the neural network, before proceeding with the next step. This data exchange is carried out using transmission control protocol (TCP) at the end of each step and may slow down the overall training procedure depending on the network connectivity data rate that is used between the workers hosting the distributed training containers.

The following configurations are used in the experiments described in this section. The deep neural network training task Ω_{k,ω_l} is executed using two containers ($k = 2$), of equal flavor. One of the following service placements is applied to the two containers. 1) The containers run in the same isolated cloudlet and optical circuits are not used. 2) Each container runs in a distinct cloudlet and the two cloudlets are connected by a lightpath routed over a fiber optics cable of a few meters. 3) Each container runs in a distinct cloudlet and the two cloudlets are connected by a lightpath routed over a fiber optics cable of 25 km. When using the super-cloudlet the data exchange at the end of each step is carried through the 10G tenant network using Fujitsu (1FINITY) Switchponder and Ciena Switchponder. Two flavors ω_1 and ω_2 are applied, i.e., $c(\omega_1) = 4$ vCPUs and $c(\omega_2) = 8$ vCPUs. The CIFAR10 dataset is divided to form batches of 32 or 128 samples. The number of steps in each epoch is then $m = 781$ (50000/32/2) and $m = 195$ (50000/128/2), respectively.

TABLE IV
EPOCH COMPLETION TIME FOR THREE SCENARIOS.

Batch Size m	**# of vCPUs per container**	**single cloudlet**	**super-cloudlet few meters**	**super-cloudlet 25km**
32	4	1075s	1092s	1095s
32	8	533s	527s	535s
128	4	1058s	1048s	1060s
128	8	501s	502s	504s

TABLE V
NETWORK UTILIZATION FOR THREE SCENARIOS.

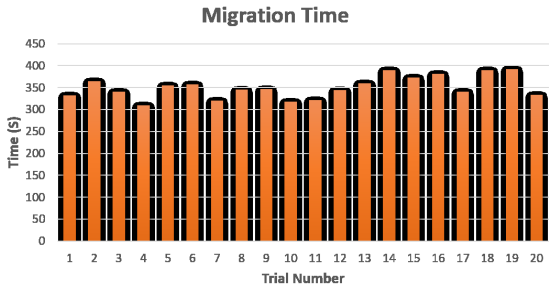
Batch Size m	**# of vCPUs per container**	**single cloudlet**	**super-cloudlet few meters**	**super-cloudlet 25km**
32	4	21.04Mbit/s	21.40Mbit/s	20.53Mbit/s
32	8	43.02Mbit/s	43.89Mbit/s	42.72Mbit/s
128	4	5.54Mbit/s	5.45Mbit/s	5.36Mbit/s
128	8	11.64Mbit/s	11.83Mbit/s	11.54Mbit/s

Tables IV and V report the epoch completion time and network utilization, respectively, for four configurations and three cloudlet settings. The number of vCPUs assigned to each container affects both epoch time and network utilization. The epoch completion time is reduced to approximately half when using 8 vCPUs compared to when using 4 vCPUs. The network utilization — defined as the tenant data rate in and out of each container — doubles when using 8 cores compared to when using 4 vCPUs, as the former has twice the computing power and completes twice as many steps per time unit (doubling the data exchanges) compared to the latter. The batch size — which determines the number of steps in each epoch — does not have a significant effect on the epoch completion time. However, it significantly affects the tenant network utilization, which is proportional to the number of data exchanges that are performed at the end of each step. Results show only minor variations across the three cloudlet settings corroborating the earlier claim that making use of

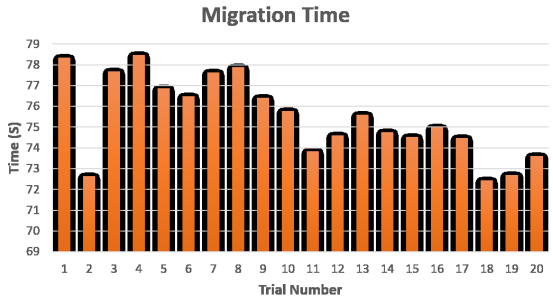
TABLE VI
MEAN OFFLOADING TIME AND VARIANCE UNDER DIFFERENT SCENARIOS.

Live VM-migration Method	30G node in 3 containers task scenario	30G node in 5 containers task scenario	1G node in 3 containers task scenario	1G node in 5 containers task scenario
Pre-copy with auto-converge	Mean: 75.4s Variance: 3.75s	Mean: 79.37s Variance: 4.2s	Mean: 351.61s Variance: 651.83s	Mean: 362.22s Variance: 583.82s
Post-copy	Mean: 83.49s Variance: 6.86s	Mean: 85.66s Variance: 7.11s	Failed	Failed

a dedicated low-latency and high-data rate optical circuit to update the tenant network in the presence of a super-cloudlet enables clusters to make use of workers in the auxiliary cloudlet without experiencing major drawbacks. Also, when increasing the length of the fiber connecting the home and auxiliairy cloudlets from a few meters to 25 km the epoch completion time increases by 1.5% or less, which is a modest performance penalty. At the same time network utilization decreases up to 4.1% due to the increased network round trip time. The training achievable accuracy is outside the scope of this paper.



(a) Migration time using nodes with 1G interface.



(b) Migration time using nodes with 30G interface.

Fig. 8. Offloading results under 5 containers task scenario with various management network rate.

C. Worker Offloading

This section focuses on the completion time of the live migration of workers (VMs) between two cloudlets. Each worker hosts one of the containers that perform the training of ResNet56 [36] as described in Section IV-B. Two methods of VM live migration are considered: pre-copy with auto-convergence and post-copy.

We also examine the effect of other factors as well. First, the distribution level of the containerized application in the COE cluster may influence the migration time of a worker. More specifically, with one container assigned to each active worker, data exchanges between every pair of workers must take place at the end of each step. These data exchanges may affect the migration time of the VM. This dependency is investigated by considering two cases: Tasks requiring 3 and 5 containers (workers), respectively. Second, the management network interface data rate too may affect the migration completion time. This dependency is investigated by considering two cases: 1G and 30G network rate. The two charts in Fig.8 report the migration completion time of 40 trials when migrating one container in a group of 5. Mean and variance of these results are reported in Table VI, along with the mean and variance results of six other system configurations.

When successful, the post-copy method requires 8% to 11% more time to complete migration compared to the pre-copy with auto-converge method. The main reason for this result is that the ML training application is a memory intensive task and the migration process must keep up with the rate of memory changes taking place in the source worker while resolving all dirty memory pages. Conversely, The pre-copy method with auto-converge throttles the CPU of the source worker and temporally slows down the worker to keep up with the memory changes, thus reducing the time that is required to complete migration.

The management network interface data rate has a significant impact on the migration completion time. By increasing the management network interface rate from 1G to 30G the migration completion time is reduced to about one fifth. The reason for this outcome is again the rate of memory change and the memory-intensive nature of the machine learning training task. A higher network data rate facilitates the copying of the memory pages to the destination in such a way that the migration procedure can keep-up with the memory changes at the source. It should also be noted that the post-copy method completes successfully only when operating with 30G, while it fails when operating with 1G. This failure is caused by the network low data rate which cannot keep up with the network related page-faults. The migration continues until it times out when the worker is completely out of service since part of it is running at the destination while other parts are still at the source. Migrating one of 5 containers takes 3% to 5% more time compared to the case in which one of 3 containers is migrated.

V. Summary

In this paper we describe a federated edge computing system implemented using commercial optical transport network equipment that is OpenROADM compliant. Service placement, auto-scaling, and offloading are performed in the federated edge system using the PROnet SDN Orchestrator, which automatically triggers these functionalities across pairs of cloudlets when needed. Once paired together to form a super-cloudlet, neighboring cloudlets can statistically share their compute resources and best handle peaks of offered load originating from their edge-supported applications. For improved system response time, a parallel resource provisioning technique is described in which workers are added to an edge compute cluster while in parallel optical circuits are being set up between the two cloudlets to form the super-cloudlet. Super-cloudlets can be set up dynamically to efficiently cope with time-variant load conditions without requiring excessive overprovisioning of resources in both compute and network domains.

Funding

This work is supported in part by NSF grants CNS-1405405, CNS-1409849, ACI-1541461, CNS-1531039, and CNS-1956357.

References

[1] A. Yousefpour *et al.*, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762118306349

[2] A. C. Baktir *et al.*, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2359–2391, Fourthquarter 2017.

[3] F. Griffiths and M. Ooi, "The fourth industrial revolution - industry 4.0 and iot [trends in future i m]," *IEEE Instrumentation Measurement Magazine*, vol. 21, no. 6, pp. 29–43, December 2018.

[4] X. Chen *et al.*, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.

[5] "Cloud Edge Computing: Beyond the Data Center," https://www.openstack.org/edge-computing/cloud-edge-computing-beyond-the-data-center/.

[6] H. T. Dinh *et al.*, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/wcm.1203

[7] M. Satyanarayanan *et al.*, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[8] F. Bonomi *et al.*, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513

[9] H. H. Pang and K. . Tan, "Authenticating query results in edge computing," in *Proceedings. 20th International Conference on Data Engineering*, April 2004, pp. 560–571.

[10] M. Cvijetic and I. Djordjevic, *Advanced optical communication systems and networks*. Artech House, 2013.

[11] "OpenROADM MSA," http://OpenROADM.org.

[12] S. Mondal *et al.*, "A novel cost optimization framework for multi-cloudlet environment over optical access networks," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1–7.

[13] S. Mondal *et al.*, "Computation offloading in optical access cloudlet networks: A game-theoretic approach," *IEEE Communications Letters*, vol. 22, no. 8, pp. 1564–1567, 2018.

[14] L. Chen and J. Xu, "Socially trusted collaborative edge computing in ultra dense networks," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3132211.3134451

[15] M. Kriushanth and L. Arockiam, "Load balancer behavior identifier (lobbi) for dynamic threshold based auto-scaling in cloud," in *2015 International Conference on Computer Communication and Informatics (ICCCI)*, 2015, pp. 1–5.

[16] R. S. Shariffdeen *et al.*, "Workload and resource aware proactive auto-scaler for paas cloud," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, 2016, pp. 11–18.

[17] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–12.

[18] "TACC's Stampede system," https://portal.tacc.utexas.edu/archives/stampede.

[19] "OpenStack," https://www.openstack.org/.

[20] B. Mirkhanzadeh *et al.*, "Demonstration of joint operation across openroadm metro network, openflow packet domain, and openstack compute domain," in *Optical Fiber Communication Conference (OFC) 2020*. Optical Society of America, 2020, p. W3C.3. [Online]. Available: http://www.osapublishing.org/abstract.cfm?URI=OFC-2020-W3C.3

[21] H. Kang *et al.*, "Container and microservice driven design for cloud infrastructure devops," in *2016 IEEE International Conference on Cloud Engineering (IC2E)*, 2016, pp. 202–211.

[22] I. M. A. Jawarneh *et al.*, "Container orchestration engines: A thorough functional and performance comparison," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[23] "Kubernetes," https://kubernetes.io/.

[24] "Docker Swram," https://docs.docker.com/engine/swarm/.

[25] "Apache Mesos," http://mesos.apache.org/.

[26] "Open Network Automation Platform (ONAP)," https://www.onap.org/.

[27] B. Mirkhanzadeh *et al.*, "An sdn-enabled multi-layer protection and restoration mechanism," *Optical Switching and Networking*, vol. 30, pp. 23 – 32, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S157342771730228X

[28] "Generic Routing Encapsulation (GRE)," https://tools.ietf.org/html/rfc2784.

[29] "Virtual eXtensible Local Area Network (VXLAN)," https://tools.ietf.org/html/rfc7348.

[30] "Transport PCE Wiki," https://wiki.opendaylight.org/view/TransportPCE:Main.

[31] "OpenStack Live VM Migration," https://docs.openstack.org/nova/latest/admin/configuring-migrations.html.

[32] B. Mirkhanzadeh *et al.*, "Demonstration of an openroadm sdn-enabled network for geo-distributed data centers," in *2019 21st International Conference on Transparent Optical Networks (ICTON)*, 2019, pp. 1–4.

[33] Y. Han *et al.*, "Convergence of edge computing and deep learning: A comprehensive survey," *CoRR*, vol. abs/1907.08349, 2019. [Online]. Available: http://arxiv.org/abs/1907.08349

[34] K. S. Chahal *et al.*, "A hitchhiker's guide on distributed training of deep neural networks," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 65 – 76, 2020.

[35] "Distributed training with TensorFlow : TensorFlow Core," https://www.tensorflow.org/guide/distributed_training.

[36] "ResNet," https://keras.io/examples/cifar10_resnet.

[37] "CIFAR10," https://www.cs.toronto.edu/~kriz/cifar.html.