Heterojen Bulut Kaynaklarının Yönetimi İçin OpenStack Genelleştirimi
A Generalization of OpenStack for Managing Heterogeneous Cloud Resources

Ahmet Erol*†, Alper Yazar*†, Ece Güran Schmidt†

† Elektrik Elektronik Mühendisliği Bölümü, ODTÜ, Ankara, Türkiye
{ahmet.erol, alper.yazar, eguran}@metu.edu.tr
* Savunma Sistem Teknolojileri, ASELSAN A.Ş., Ankara, Türkiye
{ahmeterol, ayazar}@aselsan.com.tr

***Özetçe* —Bu bildiride OpenStack bulut kaynak yönetimi yazılımının sunuculardaki standart kaynakların dışında ki donanım kaynaklarını da yönetebilecek şekilde genelleştirimi anlatılmaktadır. Bu amaçla kaynak veri yapıları güncellenmiş ve hesaplama düğümünde çalışan OpenStack Nova bileşeni farklı donanım platformlarında ve işletim sisteminden bağımsız çalışabilecek şekilde yeniden yazılmıştır.**

Anahtar Kelimeler—Bulut bilişim, sanallaştırma, OpenStack

***Abstract*—This paper describes the generalization of OpenStack cloud resource management software to manage hardware resources other than the standard resources on the servers. To this end, OpenStack resource data structure is updated and the Nova project, which runs on the compute node, is rewritten so that it can run on different hardware platforms without depending on the operating system.**

Keywords—Cloud computing, virtualization, OpenStack

I. GİRİŞ

OpenStack bulut bilişim sistemlerinde fiziksel kaynakların sanal makineler (Virtual Machines-VM) halinde kullanıcılara atamasını yapmak amacıyla pek çok büyük bulut kullanıcısı tarafından tercih edilen açık kaynaklı bir yazılımdır [1]. Kaynakların yönetimini OpenStack Nova projesi yapmaktadır. Mevcut Nova gerçekleştirimi CPU, bellek ve hafıza gibi geleneksel hesaplama kaynakları ile sınırlıdır. Buna ek olarak, OpenStack sunucularda çalışmak üzere tasarlandığı için Nova belli işletim sistemleri ve hipervizör (hypervisor) yazılımları ile uyumludur.

Yeni kullanım ihtiyaçlarına uygun olarak bulut bilişim sistemleri donanım hızlandırıcılarla, GPU (Graphics Processing Unit), TPU (Tensor Processing Unit), IoT donanımları ile entegre edilmektedir [2]. Bu farklı donanım platformları standart bulut sunucularında kullanılan işletim sistemi ve hypervisor yazılımları ile uyumlu olmayabilir.

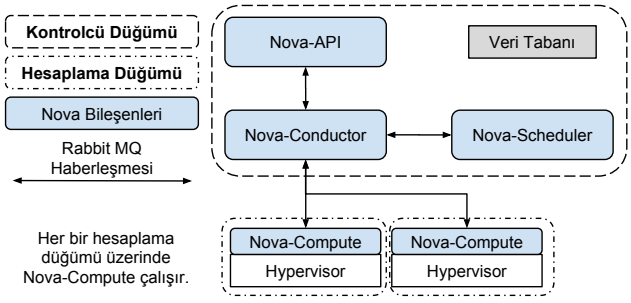
Bu bildiride OpenStack bünyesindeki ana kaynak yönetimi projesi olan Nova modülünü herhangi bir hesaplama kaynağını destekleyecek şekilde genelleştiren Nova-G, Nova-**G**enelleştirilmiş önerilmektedir. Bu amaçla Nova projesi yeni kaynak tiplerini tanıyacak hale getirilmiş ve hesaplama düğümleri üzerinde çalışan Nova bileşeni sadeleştirilerek tekrar yazılmıştır. Yapılan gerçekleştirim Python dilinde ve işletim sisteminden bağımsızdır. Bu şekilde farklı donanım platformları desteklenebilmektedir.

978-1-7281-1904-5/19/$31.00 ©2019 IEEE

II. OPENSTACK BULUT YÖNETİM YAZILIMI VE ÖNCEKİ ÇALIŞMALAR

Bulut bilişim sistemlerinde OpenStack yazılımı, sanal makineler (VM) yolu ile kullanıcılara atanan hesaplama, depolama ve ağ kaynaklarını denetler. Bu VM'ler, CPU, bellek, disk ve ağ oluşturma [3] gibi kaynakların önceden tanımlanmış konfigürasyonlarına (*flavors*) göre yapılandırılır. OpenStack, proje (*projects*) adı verilen birçok modülden oluşur.

Bu bildirinin ana konusu olan **Nova**, hypervisor ve sanallaştırma yolu ile hesaplama düğümlerini oluşturan OpenStack projesidir. Nova alt bileşenleri ve aralarındaki haberleşme yapısı Şekil 1'de gösterilmektedir.



Şekil 1: OpenStack Nova Blokları ve Mesajlaşması

Nova-API VM oluşturma, flavor güncelleme, hypervisor özelliklerini listeleme gibi Nova hizmetlerine erişime izin verir. **Nova-Conductor**, Nova'nın faaliyetlerini yöneten ana bölümüdür. Kontrolcü düğümü üzerinde çalışır ve Nova'nın

diğer tüm bileşenleri Nova-Conductor ile iletişim kurar. **Nova-Compute**, hesaplama düğümlerinde çalışan istemci yazılımıdır. **Nova-Scheduler**, kullanıcının seçtiği flavor'la tanımlanan VM'yi çalıştıracak hesaplama düğümünü seçer. **Flavor veri yapısı** `sqlalchemy api` modeline uygun bir veri yapısı ile tanımlanır. Bu veri yapısına uyumlu olarak hesaplama düğümlerindeki mevcut kaynakları tutan ve kontrolcü düğümü üzerinde tutulan bir **Veri tabanı** (Database-DB) bulunmaktadır.

Nova aşağıdaki temel adımlarla yeni bir VM oluşturur: (1) Kullanıcının seçtiği flavor'da VM isteği Nova-API üzerinden Nova-Conductor'a iletilir. (2) Nova-Conductor bu isteği biçimlendirerek Nova-Scheduler'a iletir. (3) Nova-Scheduler DB'ye erişerek hesaplama düğümlerinden seçilen flavor'a uygun mevcut kaynağı olan bir hesaplama düğümünü seçer. (4) Nova-Conductor, seçilen düğüm üzerindeki Nova-Compute'a VM bilgisini yollar. (5) VM düğüm üzerindeki hypervisor tarafından başlatılır. (6) Nova-Conductor DB'yi VM'e ayırılan kaynaklar ile günceller.

Bu süreçte bilgi akışı Advanced Message Queue Protocol (AMQP) gerçekleyen **RabbitMQ** (Rabbit Message Queue) yapısı üzerinden RPC (Remote Procedure Call-Uzaktan Prosedür Çağırımı) ile gerçekleştirilir. Bu yöntemle OpenStack Projeleri ve bileşenleri gelen mesajları çözerek (decode) mesaj içinde olabilecek işlev çağrılarını gerçekleştirirler.

[4]'te donanım hızlandırıcılı düğümler için sanallaştırmaya odaklanılmış, OpenStack tarafında yapılması gereken değişiklikler hakkında bilgi verilmemiştir. [5]'te ise Nova'nın mevcut `extra_spec` alanı kullanılarak GPU'ya özel bir gerçekleştirim yapılmıştır. [6]'da OpenStack ağ servislerini donanım anahtarlarına aktararak ağ çalışmasını hızlandırmış ve Nova'nın bu kullanıma erişimini sağlamıştır. [7]'de OpenStack'e IoTronic isimli yeni bir servis ekleyerek IoT sistemleri için algılama servisi (Sensing as a Servis) gerçekleştirmiştir.

III. NOVA-G

OpenStack genelleştirimi kapsamında kontrolcü düğümü üzerinde çalışan Nova-Conductor bileşenlerinde değişikler yapılmıştır ve hesaplama düğümünde çalışacak olan Nova-G modülü, Nova-compute yerini alacak şekilde oluşturulmuştur.

A. Nova-Conductor Değişiklikleri

Veri tabanı: OpenStack kaynak veri tabanı MySQL kullanılarak gerçekleştirilmiştir. Tablo I mevcut OpenStack girdilerinden örnekler ile birlikte Nova-G için istenen kaynakların nasıl eklendiğini göstermektedir. Bu tabloya istenen yeni kaynakların mevcut ve kullanılmış miktarları eklenir. Örnek olarak `resource_g1` yerine `fpga`, `resource_g2` yerine `gpu` yazılarak mevcut FPGA bölge ve GPU çekirdek miktarları tamsayılı değerler olarak belirtilir. Bu kaynakların kullanım durumları da `fpga_used` ve `gpu_used` alanlarında gösterilir. İstenilen sayıda yeni kaynak tipi eklenebilir. Bu gösterim ile [2]'de önerildiği gibi üzerinde işlemci olan FPGA kartının doğrudan bulut düğümü olarak işlev gördüğü durumlarda `vcpus` değeri 0, `fpga` değeri mevcut yeniden yapılandırılabilir bölge sayısı olarak belirtilir. Bu şekilde herhangi bir kaynak konfigürasyonu tanımlanabilir.

Flavor Veri Yapısı: Kullanıcı VM istekleri Nova-API üzerinden Flavor veri yapısı yolu ile gelmektedir. Yeni tip

TABLO I: Nova-G veri tabanı

Field	Type	Field	Type
id	Integer	service_id	Integer
vcpus	Integer	vcpus_used	Integer
memory_mb	Integer	memory_mb_used	Integer
local_gb	Integer	local_gb_used	Integer
hypervisor_type	Text	hypervisor_version	Text
cpu_info	Text		
resource_g1	Integer	resource_g1_used	Integer
resource_g2	Integer	resource_g2_used	Integer

kaynakları içeren VM'lerin kullanıcılar tarafından istenebilmesi için veri yapısı `sqlalchemy api model` sınıfında uygun şekilde değiştirilmiştir. `resource_g` için yeni bir satır eklenmiş olup yapılan değişiklikler sonucunda oluşan yapı Listeleme 1 de gösterilmiştir. Veri tabanındaki "Flavor" tabloları bu veri yapısı ile güncellenmiştir.

Listeleme 1: Flavor veri yapısında yapılan değişiklikler sonrası `sqlalchemy` modeli

```
class Flavors(API_BASE):
__tablename__ = 'flavors'
__table_args__ = (
schema.UniqueConstraint("flavorid",
    name="uniq_flavors0flavorid"),
schema.UniqueConstraint("name",
    name="uniq_flavors0name"))

id = Column(Integer, primary_key=True)
name = Column(String(255), nullable=False)
memory_mb = Column(Integer, nullable=False)
vcpus = Column(Integer, nullable=False)
resource_g1 = Column(Integer, nullable=False)
resource_g2 = Column(Integer, nullable=False)
root_gb = Column(Integer)
ephemeral_gb = Column(Integer)
flavorid = Column(String(255), nullable=False)
swap = Column(Integer, nullable=False, default=0)
vcpu_weight = Column(Integer)
disabled = Column(Boolean, default=False)
is_public = Column(Boolean, default=True)
```

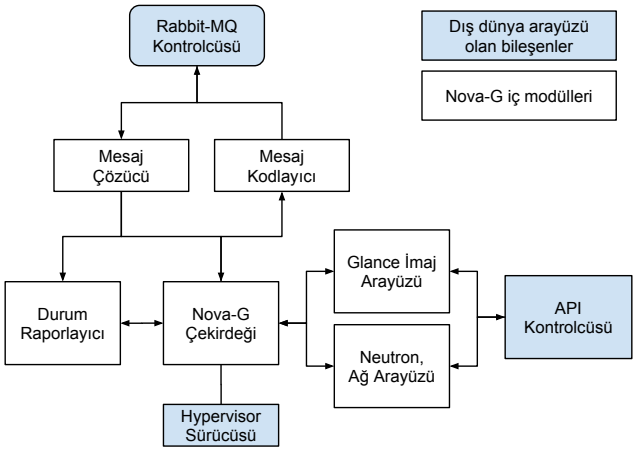
Nova-Scheduler: Nova-G'deki Nova-Scheduler yukarıda anlatıldığı şekilde genelleştirilmiş olan "Flavor"'ları kullanarak en uygun hesaplama düğümünü seçer. `resource_g1`, `resource_g2` alanını, `vcpus` (sanal işlemciler) gibi standart kaynak tipleri ile aynı şekilde filtreleme algoritmalarında kullanır.

B. Yeni Nova-Compute: Nova-G

OpenStack ile yönetmek istediğimiz kaynaklar ilgili hesaplama düğümünde çalışan Nova-Compute ve hypervisor yolu ile sanallaştırılarak kullanıcılara atanmaktadır. Standart Nova-Compute, sunucu üzerinde çalışmak üzere yazılmış ve pek çok karmaşık bileşeni olan bir yazılımdır. Nova-G [2]'de önerildiği gibi hızlandırılmış bulutlarda tek başına çalışan ve üzerinde işlemci olan FPGA kartları ya da IoT donanımı gibi özel platform kaynaklarını kapsayacak şekilde tasarlanmıştır. Bu farklı platformlara uyum sağlayabilmek için Nova-Compute yazılımının sadece gerekli bileşenlerini içererek ve işletim sisteminden bağımsız şekilde Nova-G yazılımı yeniden Python dilinde yazılmıştır. Nova-Compute'a benzer şekilde oluşturulan Nova-G blok mimarisi Şekil 2'de görülmektedir.

RabbitMQ: Nova-G modülünün işlevini yerine getirebilmesi için modülün, Nova-Conductor ve Neutron gibi diğer OpenStack projeleri ile haberleşmesi gerekmektedir.

OpenStack mimarisinde modül arası haberleşme için kullanılan RabbitMQ (Message Queue), geliştirilen Nova-G yazılımında da kullanılmıştır. Bu sebeple Nova-G kurulan hesaplama düğümlerine RabbitMQ kurulması gerekektedir. Nova-Conductor'da yukarıda anlatıldığı şekilde değişmiş olan Veri tabanı ve Flavor veri yapısı sayesinde RabbitMQ üzerinden alınıp gönderilen mesaj içerikleri yeni kaynak bilgilerini içermektedir. Nova-G, RabbitMQ Mesaj Çözücü/Kodlayıcı blokları bu yeni bilgilere uygun olarak yazılmıştır. Çözümlenmiş mesajlar Nova-G tarafından alındıktan sonra, mesaj tipine göre gerekli işlemler gerçekleştirilir.



Şekil 2: Nova-G mantıksal şeması

OpenStack Proje API destekleri: Nova-G diğer bir önemli özelliği ise Neutron ve Glance gibi diğer OpenStackprojelerinin API arayüzlerini kullanabilmesidir. Glance sanal makinelerin imajlarını yönettiği gibi sunduğu API sayesinde diğer modüllerin de imajları yönetmesine olanak sağlamaktadır. Nova-G Glance API arayüzünü kullanarak kullanıcının istediği yeni tipte kaynaklar eklenmiş sanal makine imajlarını alıp, istenilen kaynağı istenilen imaj ile ayağa kaldırabilmektedir. Neutron API arayüzünü kullanarak sanal makinelerin ağ bağlantılarını da kontrol edebilmektedir.

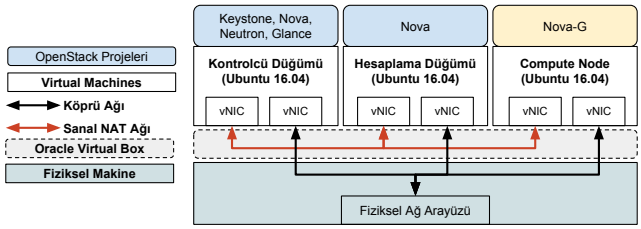
Hypervisor destekleri: Nova-Compute sunucularda çalışabilen tanımlı hypervisorlarla çalışmaktadır [8]. Nova-G çalıştıracak düğümlerin standart sunucu olmayabileceği varsayılarak FPGA gibi donanımları sanallaştırabilecek özel hypervisor benzeri yazılımlarla arayüz sağlamak için Hypervisor Driver (sürücü) bloğu yazılmıştır. Bu blok genelleştirilmiş kaynak tipine bağlı olmadan özelliştirilmiş hypervisor'ların Nova-G modülüne eklenmesini sağlamaktadır. Nova-G çekirdeği bir sanal makinenin çalışabilmesi için gerekli olan bütün bilgileri topladıktan sonra hypervisor sürücüsüne iletir. Hypervisor sürücüsü kaynak türüne göre özelleştirilmiş hypervisorı kontrol ederek ilgili sanal makinenin çalışmasını başlatır. Hypervisor sürücüsü çalıştırılmış olan sanal makineler hakkında topladığı bilgiler Nova-G çekirdeğine iletir. Nova-G çekirdeği ise bu bilgileri sanal makinenin durumunu raporlamak amacıyla Nova-Conductor modülüne iletir. Bu mesaj akışı Bölüm II'de belirtilen standart OpenStack çalışmasıyla uyumludur.

IV. Değerlendirme

Geliştirilen Nova-G yazılımı bir test düzeneğinde test edilmiştir. Bulut fiziksel düğümlerini temsil etmek üzere tek bir bilgisayar üzerinde, sanal makineler oluşturularak kontrolcü düğümü ve hesaplama düğümleri gerçeklenmiştir. Böylece düğümlerin yönetimde kolaylık sağlanış ve aralarındaki ağ sanal NAT ağı ile oluşturulabilmiştir. Kullanılan bilgisayarda Intel i7-7500U işlemci, 16 GB bellek ve Windows 10 işletim sistemi bulunmaktadır. Sanallaştırma aracı olarak Oracle VirtualBox kullanılmıştır.

A. Test Düzeneğinin Genel Yapısı

Kontrolcü düğümü ve hesaplama düğümü Ubuntu 16.04 LTS işletim sistemi ile oluşturulmuştur. Bütün oluşturulan düğümler iki adet sanal ağ arayüzüne sahiptir. İlk ağ arayüzü OpenStack modüllerinin yönetimi için, ikinci arayüz ise düğümlerin Internete bağlanmasını sağlamak amacıyla kullanılmaktadır. Test düzeneğinin detaylı şeması Şekil 3'te gösterilmiştir.



Şekil 3: Test düzeneği şeması

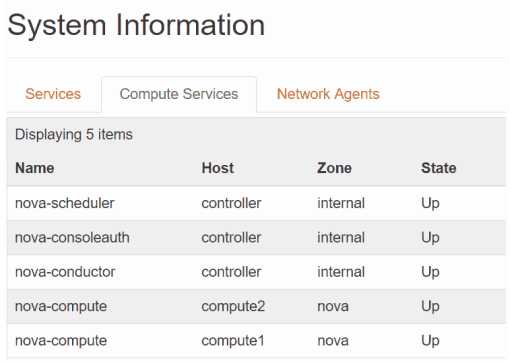
Kontrolcü düğümüne OpenStack projelerinden Keystone, Nova, Neutron ve Glance kurulmuş ve fonksiyonel olarak çalışacak şekilde ayarlanmıştır. Birinci hesaplama düğümü standart bir sunucu olarak Nova-Compute ile kurulmuştur. Diğer hesaplama düğümünde Nova-G yazılımı RabbitMQ ile birlikte kurulmuştur. OpenStack yazılımı iki farklı düğüm tipini de (Nova ve Nova-G) tanıyarak çalışmaya başlamıştır.

B. Yapılan Testler

Test 1: Bütün OpenStack haberleşmeleri ağ üzerinden yapıldığı için ilk olarak bütün düğümlerin birbiriyle bağlantısının olduğu doğrulanmıştır. Gecikme testi sanal NAT ağını kullanan arayüzler arasında yapılmıştır. Ortalama ping zamanı 0.335 ms olarak ölçülmüştür. Bu gecikme değeri bütün haberleşme gecikmelerinde referans değeri olarak kullanılmıştır.

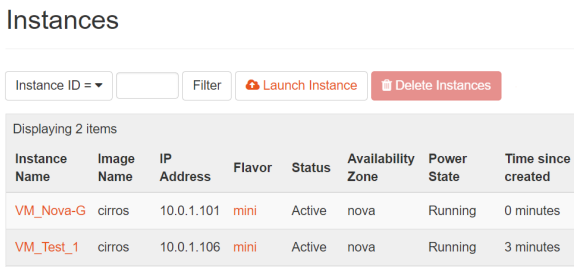
Test 2: OpenStack servislerinin çalıştığı ve birbirleriyle doğru bir şekilde haberleştiği test edilmiştir. Bu testte OpenStack Horizon modülü kullanılmıştır. OpenStack Horizon modülü sayesinde bütün OpenStack yazılımlarının durumu Şekil 4'te gösterildiği izlenebilmekte ve takip edilebilmektedir. Standart OpenStack projelerinin yanı sıra geliştirilmiş olan Nova-G modülünün de başarılı bir şekilde çalıştığı gözlemlenmiştir. `compute1` referanslı hesaplama düğümü üzerinde Nova-G yazılımı çalışmakta iken, `compute2` referanslı hesaplama düğümü üzerinde standart Nova yazılımı çalışmaktadır. İki yazılım da OpenStack sistemi tarafından eşlenik olarak görülmüş olup `nova-compute` olarak adlandırılmıştır. Bu test sayesinde geliştirilen sistemin OpenStack ile birebir uyumlu

olduğu doğrulanmıştır. Ayrıca Nova-G modülünün durumunu sürekli güncellediği, `state` olarak belirtilen durumun `up` olduğu görülerek doğrulanmıştır.



Şekil 4: Horizon üzerinden çalışan servislerin test edilmesi

Test 3: Kullanıcın yaptığı sanal makine isteklerinin başlatılması hem orjinal Nova ve hem de geliştirilen Nova-G üzerinde test edilmiştir. Kullanıcının yapmış olduğu isteğin başarılı bir şekilde yerine getirildiği Şekil 5'te gösterildiği gibi doğrulanmıştır. Nova-G modülünün haberleştiği, özelleştirilmiş hypervisoru temsil edecek test amaçlı bir hypervisor geliştirilmiş ve kullanılmıştır. `VM_Test_1` adlı sanal makine standart Nova-Compute koşan `compute2` adlı hesaplama düğümünde çalışmaktadır. `VM_Nova-G` adlı sanal makine geliştirilen Nova-G modülünün kullanıldığı `compute1` adlı hesaplama düğümünde çalışmaktadır. Bu test sonucunda Nova-G yazılımının başarılı bir şekilde istekleri aldığı ve sanal makineyi başlatabildiği doğrulanmıştır.



Şekil 5: Sanal makine isteklerinin yerine getirilmesi

Test 4: Nova-G modülünün Rabbit MQ kullanarak diğer Nova modülleri ile haberleşme performansı test edilmiştir. Yapılan ilk testte Nova-G modülü üzerinde çalıştığı hesaplama düğümünün bilgilerini göndermesi sırasındaki gecikme ölçülmüştür. Her yüz 100 ms'de bir hesaplama düğümünün durumunu güncellemiş her mesajın gecikme süresi ölçülmüştür. Düğümler arası ping gecikmesinin 0.335 ms olduğu test düzeneğinde 0.5 ms mesaj gönderme gecikmesi geliştirilen sistemin ekstra yük getirmeden hızlı bir şekilde çalıştığını göstermektedir.

Test 5: Nova-G, Nova-Conductor modülü üzerindeki `Service` objelerinin `get_by_uuid` methodu kullanılarak RPC isteği yapılmıştır. 100 adet istek 100 ms zaman aralığı ile yapılmış ve tepki süresi ölçülmüştür. Yapılan ölçümlere ait istatistikler Tablo II'de özetlenmiştir. RPC sürelerinin kabul edilebilir olduğu görülmüştür.

TABLO II: RPC isteği gecikme zamanı istatistikleri

Durum güncellemesi	**Zaman (ms)**
Minumum Gecikme	0.4
Maksimum Gecikme	1.2
Ortalama Gecikme	0.5
RPC İsteği	**Zaman (ms)**
Minumum Gecikme	9
Maksimum Gecikme	30
Ortalama Gecikme	14

Nova-G modülünün bellek kullanımı normal işlevi sırasında Python `psutil` kütüphanesi ile ölçülmüştür. Ortalama 33.4 MB bellek kullanıma sahiptir.

V. Sonuç

Bu bildiride sunulan Nova-G yazılımı çok kullanılan OpenStack yazılımını farklı kaynak tiplerini destekleyecek şekilde genişletmektedir. Nova-G diğer OpenStack projeleri ile uyumlu çalışarak, hesaplama düğümünün son durumunu kaynak tipine göre raporlayabilir, genel bir hypervisor sürücüsü sayesinde sanallaştırılmış farklı kaynak tiplerini kullanabilir, Nova ile benzer şekilde RPC istekleri yapabilir ve cevap verebilir, aldığı imaj ile başarılı bir şekilde farklı kaynak tiplerine sahip sanal makineler oluşturabilir, durumlarını raporlayabilir, ağ konfigürasyonlarını OpenStack Neutron projesi üzerinden yapabilir. Nova-G işletim sisteminden bağımsız olarak RabbitMQ destekleyen her platformda çalışabilir. Nova-G düşük donanım gereksinimleri ve düşük RAM kullanımı ile çalışabilmektedir.

Teşekkür

Bu çalışma, 117E667-117E668 nolu proje kapsamında TÜBİTAK tarafından desteklenmektedir. Yazarlar desteklerinden dolayı TÜBİTAK'a ve ASELSAN A.Ş.'ye teşekkür eder.

Kaynaklar

[1] "Openstack user stories," https://www.openstack.org/user-stories/, accessed: 2019-02-02.

[2] A. Yazar, A. Erol, and E. G. Schmidt, "Accloud (accelerated cloud): A novel fpga-accelerated cloud archictecture," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*. IEEE, 2018, pp. 1–4.

[3] "Openstack docs: Rocky," https://docs.openstack.org/rocky/, accessed: 2019-02-02.

[4] N. Tarafdar, T. Lin, E. Fukuda, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "Enabling flexible network fpga clusters in a heterogeneous cloud data center," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 237–246.

[5] S. Crago, K. Dunn, P. Eads, L. Hochstein, D. Kang, M. Kang, D. Modium, K. Singh, J. Suh, and J. P. Walters, "Heterogeneous cloud computing," in *2011 IEEE International Conference on Cluster Computing*, Sep. 2011, pp. 378–385.

[6] L. Phan and K. Liu, "Openstack network acceleration scheme for datacenter intelligent applications," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, July 2018, pp. 962–965.

[7] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito, "Stack4things: a sensing-and-actuation-as-a-service framework for iot and cloud integration," *Annals of Telecommunications*, vol. 72, no. 1, pp. 53–70, Feb 2017.

[8] "Openstack hypervisors," https://docs.openstack.org/ocata/config-reference/compute/hypervisors.html, accessed: 2019-02-02.