OpenStack Generalization for Hardware Accelerated Clouds

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

† Department of Electrical and Electronics Engineering, METU, Ankara, Turkey
{ahmet.erol, alper.yazar, eguran}@metu.edu.tr
* Defence Systems Technologies, ASELSAN, Ankara, Turkey
{ahmeterol, ayazar}@aselsan.com.tr

***Abstract*—OpenStack is a widely used management tool for cloud computing which is designed to work on servers and allocate standard computing resources such as CPU, memory or disk. The current trend for integrating different hardware accelerators such as FPGAs and GPUs in the cloud requires managing these heterogeneous resources. In this paper, we propose a generalization for OpenStack Nova project which extends the relevant data structures to include these new resources. More importantly, we present a new lightweight Nova Compute module that we call Nova-G Compute. Nova-G Compute is suitable to work with different hardware platforms and can communicate with the rest of the OpenStack Projects. We implement a hypervisor-like software to enable Nova-G Compute accessing the FPGA resources. We perform experimental evaluation of Nova-G Compute using the known and used OpenStack benchmarking tool Rally. Our results show that Nova-G Compute works as desired without any reduced performance compared to standard Nova.**

***Index Terms*—Cloud computing, virtualization, OpenStack, FPGA, Nova, Rally**

I. Introduction

Cloud computing has become a popular computing model in the recent years as it exploits the economies of scale for efficient use of computing resources. The data centers of today are mostly cloud based with virtualized servers to provide on-demand scalability and flexibility of the available resources such as CPU, memory, data storage and network bandwidth. A cloud data center provider may offer Infrastructure as a Service (IaaS), where the user gets a virtual machine (VM) with processing, memory, storage and networking resources, which can be installed with any desired operating system and software. Differently, Platform as a Service (PaaS) commonly provides a ready environment with operating system, programming language execution environment, database and web server for developers to test and deploy their programs and applications. Finally in Software as a Service (SaaS), the user only accesses the provided application for example via a web browser without any control of the underlying infrastructure.

OpenStack is an open source software that is preferred by many large cloud providers [1] to assign physical resources to users in the form of Virtual Machines (VM) in cloud computing systems. OpenStack is composed of a number of projects with different functionalities such as authentication, network management and image services. The actual managing of resources by means of virtualization is carried out by the OpenStack Nova project. Nova has two components called Nova Conductor and Nova Compute. A controller node in the managed cloud runs Nova Conductor and each server that is provisioned for VMs runs Nova Compute.

The slowdown of Moore's law and the increased data and problem sizes together with the development of high performance programmable hardware platforms such as FPGAs increase the popularity of hardware accelerators. Hardware accelerators can provide better performance and less energy consumption depending on the problem properties and size [2]. On one hand, these different hardware platforms may not be compatible with the operating system and hypervisor software used on standard cloud servers. Furthermore, their processing capabilities may be more limited and they might not be able to run OpenStack in a scalable and high performance manner. On the other hand, integrating hardware resources in the cloud based data center should be seamless, together with virtualization and dynamic resource allocation capabilities.

OpenStack is designed to work in cloud data centers with conventional servers. To this end, the current Nova implementation is limited to traditional computing resources such as CPU, memory and disk. Furthermore, Nova is only compatible with certain operating systems and hypervisor software.

The first contribution of this paper is a new lightweight project that we call Nova-G Compute which is designed to replace the standard Nova Compute for such heterogeneous hardware platforms. Nova-G Compute can work with the standard OpenStack projects by sending and receiving messages in the correct format. The implementation of Nova-G is in Python language and is independent of the operating system. In this way, different hardware platforms can be supported. The second contribution is extending the data structures of Nova Controller with the generalized resource types to work with Nova-G Compute. Different than the previous work, new resource types are defined at the same level with conventional server resources which enables using the standard Nova Schedulers to allocate the available resources to the VM requests. Nova-G Compute includes a generalized hypervisor driver

This work was supported by the Scientific and Research Council of Turkey (TUBITAK) [Project Code 117E667-117E668].

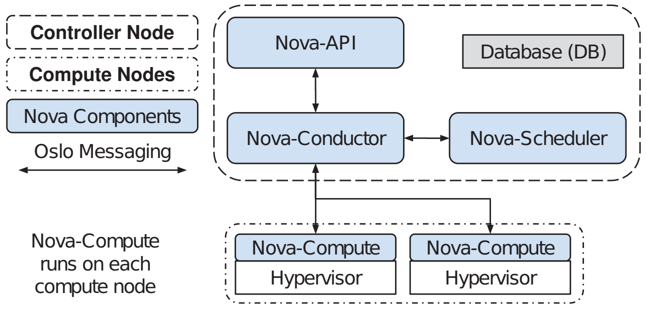


Fig. 1. Detailed Nova block diagram.

to access the available resources on the respective hardware similar to standard Nova Compute. To this end, we implement FPGAvisor software that functions similar to a hypervisor for FPGA as an example for allocation of non-standard hardware resources.

We demonstrate the functionality and performance of Nova-G Compute with a number of experiments including tests with Rally Tool which is the OpenStack framework for performance analysis and benchmarking. Our results show that Nova-G Compute can work seamlessly with OpenStack and can boot VMs as desired without any performance decrease compared to Nova Compute.

II. OpenStack and Heterogeneous Cloud Computing

A. OpenStack

OpenStack is a very popular cloud resource management tool with yearly increasing revenue in the market [3]. It controls compute, storage, and networking resources of the cloud by provisioning virtual machines (VM) [4]. These VMs are configured according to predefined *flavors* which quantify the amount of resources of the available types such as CPU, memory, disk and networking. An administrative user can create, edit, and delete flavors.

Main control functions of OpenStack run on a *controller node* whereas clients run on the *compute nodes* that are managed. OpenStack consists of many modules that are called *projects*.

Nova is the OpenStack project that enables provisioning of compute nodes through virtualization technologies such as hypervisors. The components of Nova are shown in Figure 1.

Nova API allows access to Nova services such as creating VMs, updating flavors, listing hypervisor properties [5]. **Nova Conductor** is the main part of Nova that manages its operations. It runs on the controller node and all other components of Nova communicate with Nova Conductor. **Nova Compute** is the client software that runs on the compute nodes. **Nova Scheduler** selects the compute node to run the requested VM that is defined by flavors according to available resources in the system. Flavor data structure is characterized by `sqlalchemy api` model. All the information about resources allocations which is defined by API models stored in the **Database (DB)** on the controller node.

Keystone provides the authentication mechanism. Most API clients must be authenticated by Keystone to make API requests. [6]. **Neutron** provides network connectivity to the VMs created by Nova. VMs have vNIC (virtual Network Interface Card) interfaces created by hypervisors. vNIC connectivity is established via networking services of Neutron [7]. Neutron provides an API to create and manage networks, create/delete ports and manage L3 services [8].

Glance is a VM image service which discovers, registers, and retrieves virtual machine images [9]. **Horizon** provides web user interfaces to other OpenStack projects for effortless cloud management activities such as creating VMs, monitoring and configuring networks. etc [10].

B. Nova VM Instantiation

OpenStack instantiates a new VM by the following operations which are mainly carried out by Nova [4].

1) User first gets authenticated by Keystone and then requests a VM of a selected flavor through Nova API which verifies and forwards the request to Nova Conductor.
2) Nova Conductor formats the request for scheduling and forwards it to Nova Scheduler.
3) Nova Scheduler filters the available resources according to the requested VM flavor and selects a compute node to run the VM.
4) Nova Conductor sends the VM information to Nova Compute on the the selected compute node.
5) Nova Compute requests IP & MAC addresses from Neutron and the disk image to run the VM from Glance. Once the everything is ready, it calls the hypervisor on the compute node to run the requested VM.
6) When the VM is successfully instantiated by hypervisor, Nova Conductor updates the DB and notifies the user.

All the information passing in the above steps among the OpenStack projects together with other communication to other components such as MySQL database, hypervisors, vNICs use Advanced Message Queue Protocol (AMQP) to communicate [11]. Oslo messaging protocol is implemented over a Rabbit Message Queue [12] to make Remote Procedure Calls (RPC). OpenStack supports a list of different hypervisors through specific drivers [13].

C. Heterogeneous Cloud Computing and OpenStack Support

Employing different hardware platforms in the cloud is a popular topic both academically and commercially. To this end, cloud computing systems are integrated with hardware accelerators realized on FPGA platforms [14], GPU (Graphics Processing Unit) [15] , TPU (Tensor Processing Unit) [16] and IoT Hardware [17]. [18] is a pioneering work that proposes a cloud-based data center architecture accelerated with reconfigurable FPGA for use in Microsoft data centers. In this study, an Altera Stratix V-based hardware accelerator card was added to each compute node which can accelerate an application running on the machine on which it is connected or can be used as a network appliance without putting a load on the

CPU. Regarding the management of the FPGA resources [19] proposes assigning FPGA reconfigurable regions in the scope of IaaS/HaaS (Hardware as a Service) by using OpenStack. However, it does not describe how FPGA resources are defined and assigned using Nova. In the follow-up work [20], virtualization of hardware accelerators is discussed without giving details on what should be changed in OpenStack and how it should be implemented. Here it is important to note that FPGA SoC platforms come with a processor [21] which enables the cloud service provider to employ stand alone FPGA cards without a server for implementing energy efficient accelerators.

The sub-field `extra_spec` of Nova data structure is used to integrate GPUs into OpenStack platform in the paper [22]. As this field is a sub-field of existing standard resource types, it cannot be used by the standard Nova Schedulers. In [23], network is accelerated by transferring Openstack network services to switch hardware and allowing Nova to access them. In [24], a new component named IoTronic is added to OpenStack as Sensing as a Service for IoT systems. OpenStack has a recent project called Cyborg [25] which is proposed as a service for managing accelerators such as FPGAs, GPUs etc. Cyborg is designed to work with Nova Compute as an agent and cannot work stand alone. Hence, it does not support heterogeneous cloud hardware that is not connected to a CPU via PCI. Moreover, effective scheduling of accelerators are not possible because it uses `extra_spec` field to define the different types of resources similar to [22].

III. Generalization of OpenStack Nova

The current OpenStack implementation is limited to manage conventional compute resources such as CPU, RAM and disk. Moreover, compute nodes are restricted to standard server hardware and few operating systems. To this end, we propose an extension of Nova data structure to accommodate new resource types. More importantly we propose a new and lightweight Nova Compute that we call Nova-G Compute which runs on any OS or hardware platform.

A. Nova Data Structure Extensions and Modifications to Support Generalized Resources

Compute Node Resource Database Extensions: MySQL is used as the main database in OpenStack environment. Nova Controller stores the information about the types of resources and their current usage in database tables. We extend this table with the new resource types and their usage values maintaining the original data structure. Table I shows a partial example with the current resource types and two new added fields. Here, `resource_g1` can represent `fpga` resources whereas `resource_g2` can represent `gpus`. The usage of these resources in the related compute node is represented as an Integer type. More resource types can be added to the database in the same manner. As we preserve the data structure, each new resource type can be chosen independently in a flavor to create new VM configurations. Generalized resources do not have to be on the same compute node

TABLE I
Extended Compute Node Resource Database.

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

with vcpus or memory. With the help of this flexibility, one can define a flavor without any cpu or memory that is only including generalized resources. Nova-G Compute module will instantiate `resource_g` type resources defined in flavor. For example if the compute node is a standalone FPGA accelerator card, a VM without vCPU and with a `fpga` can be instantiated.

Flavor Data Structure Extensions: Nova Controller represents the flavors for VMs using sqlaclhemy models. To this end, sqlalchemy api model should also be modified by adding `resource_g` fields so that applications which are trying to connect database can use the new resource types. The modified model is shown in the Listing 1.

Listing 1. Extended Flavors sqlalchemy model after modifications

```
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)
rxtx_factor = Column(Float, default=1)
vcpu_weight = Column(Integer)
disabled = Column(Boolean, default=False)
is_public = Column(Boolean, default=True)
```

Nova Scheduler Operation with Resource Extensions: Nova scheduler uses these extended flavors to choose suitable compute nodes for the new VMs that are requested. Therefore, adding `resource_g` fields at the same level as the conventional resources such as vcpus give lots of flexibility for scheduling algorithms. All of the the existing filtering and weighting algorithms of Nova Scheduler can be extended to include `resource_g` or new filtering algorithms can be added to leverage efficiency of scheduling algorithms.

B. Nova-G Compute

OpenStack provides virtualized compute resources by Nova Compute which works on compute nodes with the help of hypervisors. Original Nova Compute software is developed to work on cloud servers and has many complex sub-systems. Nova-G Compute is designed to replace Nova Compute and work on standalone FPGA SoCs with CPU or any other customized hardware accelerators in the cloud such as the architecture proposed in [26]. To this end, we develop a lightweight Nova-G Compute that seamlessly works with other OpenStack projects by correctly generating messages as well as correctly parsing the received messages and taking the necessary actions. Nova-G Compute does not have any external software dependencies therefore it is meant to work on any operating system. Nova-G Compute is developed from scratch in Python 2.7 language in a similar structure to Nova Compute as seen in the the block diagram that is shown in the Figure 2.

Nova-G Compute communicates with Nova Conductor and Neutron during VM instantiation in the same steps as explained in Section II-B. To this end, we use Rabbit Message Queue as in standard Nova implementation. Nova-G Compute Encoder module properly formats the messages and sends them to the Rabbit Message Queue. Similarly the Decoder module receives the messages from the Rabbit Message Queue and parses them properly. Then, the required actions are taken by Nova-G Compute according to the message contents. The standard OpenStack messages are supported together with the extensions of the resource database and flavor data structures as explained in Section III-A. OpenStack communication generally is based on RPC. Nova-G Compute core module can respond these RPC requests and can make RPC requests to the other OpenStack components.

OpenStack API Support: Nova-G Compute module can use the APIs of other OpenStack components. Nova-G Compute uses Glance API to retrieve the required VM images. Moreover, it takes advantage of Neutron API to control the OpenStack network.

Hypervisor driver: Standard Nova Compute supports predefined hypervisors which work on standard server hardware [13]. Since Nova-G Compute is designed to support non-standard hardware, appropriate support is required for the virtualization of this hardware with custom hypervisors. To this end, a hypervisor driver module is developed to abstract the Nova-G Compute implementation from the custom hypervisor depending on the type of `resource_g`. Nova-G Compute core gathers all the information and provides them to the hypervisor driver. The Hypervisor driver gives commands to the custom hypervisor for the specific hardware type to instantiate the requested VM. Moreover, it gets the VM status information from the custom hypervisor.

State Reporter: Nova-G Compute core collects the VM Status information and reports the current state of the compute node to the Nova Conductor using the State Reporter Module. These messages are compatible with the current OpenStack implementation.

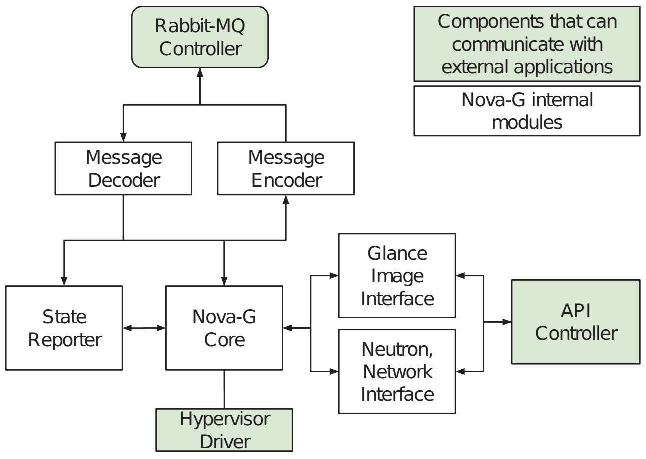


Fig. 2. Nova-G logical block diagram.

C. FPGAvisor

Nova-G Compute requires a custom hypervisor that can communicate with hypervisor drivers depending on the `resource_g` which could be any hardware component. In this work, we select FPGA as the generalized resource and develop a custom hypervisor that we call FPGAvisor to demonstrate the capabilities of our proposed extensions and Nova-G Compute. Moreover, each FPGA is not considered as single resource. FPGAs are divided into re-configurable regions. Individual re-configurable regions of FPGAs are regarded as single `resource_g` unit. Each region can be separately programmed by FPGAvisor.

FPGAvisor is controlled by the hypervisor driver of Nova-G Compute. Nova-G Compute provides the IP and MAC addresses and the image for the FPGA reconfigurable region to FPGAvisor through the hypervisor driver. Afterwards, FPGAvisor programs the selected region with the FPGA image supplied by Glance and assigns the IP and MAC addresses for the specific reconfigurable region. FPGAvisor collects data from the reconfigurable regions to provide Nova-G state reporter. The data includes utilization of FPGA reconfigurable regions, power consumption of FPGA and network traffic. This collected data are used by other OpenStack components such as Nova Scheduler for effective scheduling algorithms.

D. Capabilities of Nova-G Compute

Nova-G Compute takes advantage of the current OpenStack structure and supports heterogeneous hardware resources. Nova-G Compute provides most of the capabilities of Nova Compute without needs for changing general OpenStack implementation. Capabilities of the Nova-G Compute are summarized below.

- Reports the current state of compute nodes depending on the resource type
- Generalized hypervisor driver interface for different resources

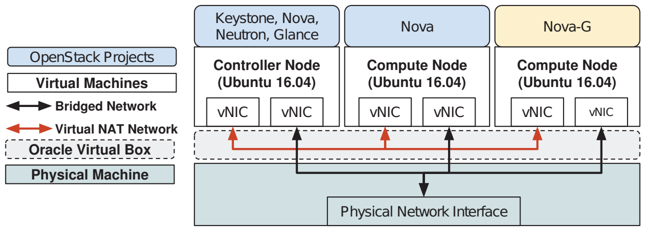

Keystone, Nova, Neutron, Glance
Nova
Nova-G
OpenStack Projects
Virtual Machines
Bridged Network
Virtual NAT Network
Oracle Virtual Box
Physical Machine
Controller Node (Ubuntu 16.04)
Compute Node (Ubuntu 16.04)
Compute Node (Ubuntu 16.04)
vNIC
vNIC
vNIC
vNIC
vNIC
vNIC
Physical Network Interface

Fig. 3. Test setup block diagram.

- Ability to make RPC request and respond to RPC requests in the same way that OpenStack does
- Successful instance initialization via given VM image
- IP & MAC configuration with Neutron project
- Instance state reporting ability
- No any other software dependencies rather than RabbitMQ
- Does not require Nova Compute to work with non-standard resources
- Lightweight implementation resulting in low RAM usage

IV. Evaluation and Results

The developed Nova-G Compute together with the extensions to support different resource types is tested on a test setup which has basic OpenStack projects such as Horizon, Nova, Neutron and Glance. All these installed projects are fully functional. The test set-up consists of a single physical computer with three virtual machines to simulate the controller node and two compute nodes. The physical computer has Intel Core i7-7500U CPU, 16GB of RAM and Windows 10 64 bit operating system. The virtualization software to create the nodes is selected as Oracle VM VirtualBox with Ubuntu 16.04 LTS operating system. VirtualBox provides flexibility with VM management, has user friendly interface and more importantly supports creating virtual NAT network for VMs.

Each virtual machine has two virtual network interfaces. The first interface is used as management network for OpenStack with pre-defined virtual NAT network. The other interfaces are used to connect to The Internet with a network bridge interface shown in Figure 3. Each node can communicate with the other nodes over the virtual NAT network with a minimum delay.

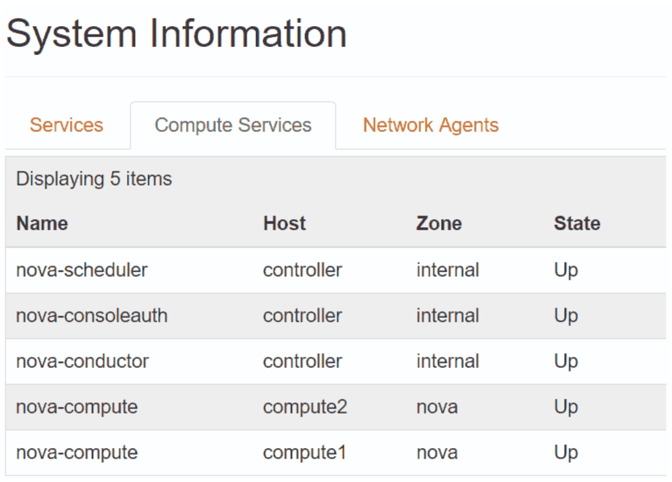
On the controller node, Keystone, Nova, Neutron and Glance OpenStack projects are installed and configured properly to work on given setup. Standard Nova Compute is installed on the first node where Nova-G Compute is installed on the second node. Nova-G only depends on the Rabbit MQ therefore Rabbit MQ is also installed separately on this node. This set-up does not include any real heterogeneous hardware and is dedicated only for the performance evaluation of the extensions for the generalizations and Nova-G Compute. Hence we modify the FPGA-visor to return with the proper response as if the FPGA reconfigurable region is programmed and ready.

A. Basic Functional Tests

Test 1: Verification of network interfaces: All OpenStack communication is over the network therefore first the connectivity between all nodes are tested. Round-trip times (RTT) between vNICs that are connected to the Virtual Network, are measured. The mean of RTT is 0.355 ms between the controller and the compute nodes. This RTT value is used as reference for other time measurements.

Test 2: Verification of Nova-G Compute compatibility: The connectivity between OpenStack services and their operation are tested. The OpenStack project, Horizon is used in this test. With the help of the Horizon Module, the state of services in OpenStack is viewed and tracked as shown in the Figure 4. Nova-G Compute is working on the compute node named as `compute1`. The original Nova Compute is active on the compute node called `compute2`. Both Nova Compute versions are identified by OpenStack system in the same manner named `nova-compute`.

Horizon shows both `compute1` and `compute2` nodes in `Up` state. Furthermore, these states are continuously updated as `Up`. This test verifies that Nova-G Compute properly operates and it is compatible with the standard OpenStack implementation.


System Information

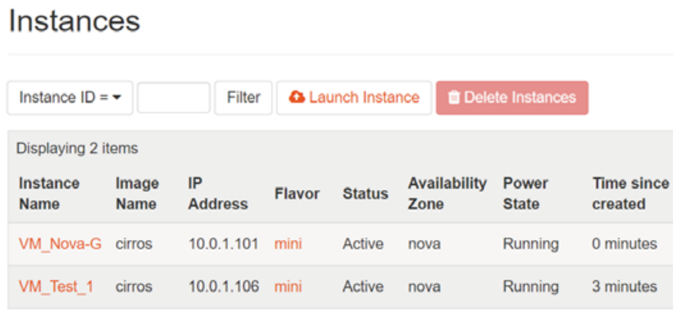
Displaying 5 items

Name	Host	Zone	State
nova-scheduler	controller	internal	Up
nova-consoleauth	controller	internal	Up
nova-conductor	controller	internal	Up
nova-compute	compute2	nova	Up
nova-compute	compute1	nova	Up

Fig. 4. Testing of OpenStack services with Horizon.

Test 3: Verification of Nova-G Compute VM instantiation: In this test, Nova-G Compute instantiates a VM while standard Nova Compute instantiates another VM. To this end, the FPGAvisor communicates with the hypervisor driver in Nova-G . Consequently, as seen in Figure 5, OpenStack Horizon shows that `VM_Nova-G` VM is working on node `compute1` which runs Nova-G Compute. In addition, `VM_Test_1` is working on node `compute2` that runs the standard Nova Compute. Hence, it is verified that Nova-G module is capable of correctly getting VM requests of the users and instantiating the VMs.

Test 4: Nova-G Compute communication latency: Every 100 ms, compute nodes update their status by sending a message to the controller node. We measure an average latency



Instances

Instance ID = ▾ | Filter | Launch Instance | Delete Instances

Displaying 2 items

Instance Name	Image Name	IP Address	Flavor	Status	Availability Zone	Power State	Time since created
VM_Nova-G	cirros	10.0.1.101	mini	Active	nova	Running	0 minutes
VM_Test_1	cirros	10.0.1.106	mini	Active	nova	Running	3 minutes

Fig. 5. VM instantiation by Nova-G.

of 0.5 ms over 100 measurements between `compute1` and the controller node when Nova-G Compute updates the status. Since the mean RTT between nodes is 0.355 ms as measured in Test 1, we conclude that Nova-G Compute works with a minumum overhead.

Nova-G Compute module uses `get_by_uuid` method of `Service` objects on Nova Conductor to make RPC requests. We measure the mean RPC request response time of Nova-G Compute over a total of 100 different RPC requests made in a time interval of 100ms. The results of measurements are summarized in Table II.

TABLE II
NOVA-G COMPUTE COMMUNICATION LATENCY.

Status Updates	Time (ms)
Minimum Latency	0.4
Maximum Latency	1.2
Average Latency	0.5
RPC Requests	**Time (ms)**
Minimum Delay	9
Maximum Delay	30
Mean Delay	14

During these tests, the memory usage of Nova-G module is measured during normal operation by Python `psutil` library. Module has an average memory usage around 33.4 MB.

B. Functional Tests Using Rally

After verifying the basic functionality of Nova-G Compute, we conduct performance tests using the benchmarking tool **Rally** [27] to test the overall operation of Nova-G Compute and compare its scalability with the standard Nova Compute. Rally is an external tool that uses OpenStack services to test cloud infrastructure with a standard testing environment. Rally is used by many companies including Intel, IBM, Huawei and Cisco to test the performance and scalability of their clouds [28]. Here we note that, our capability of using Rally for Nova-G Compute tests is an important indicator to show that Nova-G Compute is well integrated to OpenStack.

Our test set-up is on a single physical machine, hence, we test Nova Compute and Nova-G independently so that they do not affect the performance of each other. Accordingly, in each test one compute node is used either with Nova Compute or Nova-G Compute. The corresponding test case block diagrams are shown in Figure 6.

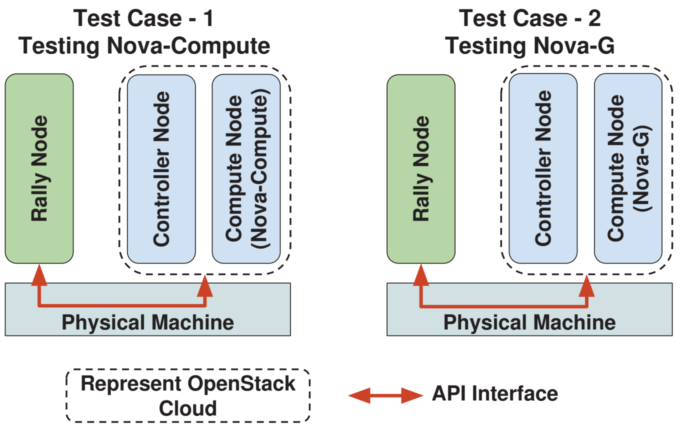
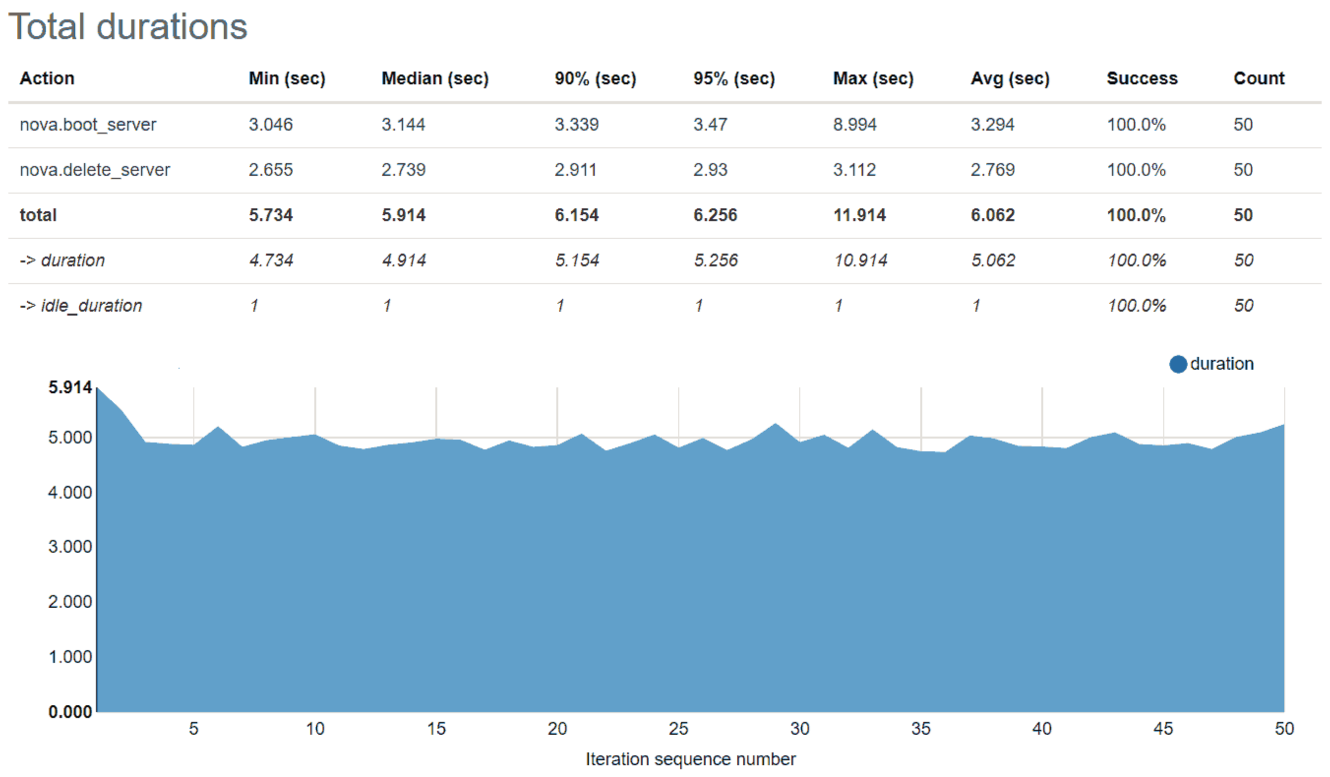


Fig. 6. Test setup used by Rally.

Test 5: Nova-G VM instantiate latency for single requests: We test the performance of Nova-G Compute in the complete operation cycle of standard VM creation, booting and deletions. To this end, Rally sends a VM creation request with a determined flavor and a VM image to the OpenStack cloud which is represented by our test set-up. Afterwards, Rally observes the state of the VM using OpenStack API. When the compute node finishes instantiating of a VM, Rally sends a delete request to the controller node to terminate the VM. Then it starts making a new VM request to the cloud. This process continues over a defined number of iterations. Figure 7 shows the booting and deletion of server times for Nova-G Compute for 50 iterations.

Test 6: Scalability of VM instantiate latency comparison: Rally is configured to make VM instantiation request to OpenStack system in these test. Rally has a configuration parameter for stating the number of concurrent VM requests. For example, if the concurrent VM request parameter is set to 2, Rally keep tracks of VMs on compute nodes and maintains 2 VMs on the compute node for testing period. For this experiment, we run 5 tests where we change the the concurrent VM request number from 1 to 5 in each test. A total of 50 VM requests are handled in each test. All tests are conducted both for original Nova Compute and Nova-G. Comparison between Nova Compute and Nova-G average VM instantiation time is shown in Figure 8. Here we note that FPGAvisor of Nova-G Compute does not execute the actual boot up actions. To this end, the hypervisor boot time of standard Nova Compute is subtracted from the total VM boot time for fair comparison. We observe that the boot time of Nova-G Compute is less than standard Nova Compute because of its lightweight implementation with essential components only.

We present the mean time figures normalized with the VM instantiation time of a single request for Nova-G Compute and standard Nova Compute respectively in Figure 9 for better demonstration of the Nova-G Compute scalability. Here we see that the VM booting time grows linearly with the number of concurrent VM requests. Hence, Nova-G Compute performs as goods as the original Nova Compute in terms of scalability.



Total durations

Action	Min (sec)	Median (sec)	90% (sec)	95% (sec)	Max (sec)	Avg (sec)	Success	Count
nova.boot_server	3.046	3.144	3.339	3.47	8.994	3.294	100.0%	50
nova.delete_server	2.655	2.739	2.911	2.93	3.112	2.769	100.0%	50
total	**5.734**	**5.914**	**6.154**	**6.256**	**11.914**	**6.062**	**100.0%**	**50**
-> duration	*4.734*	*4.914*	*5.154*	*5.256*	*10.914*	*5.062*	*100.0%*	*50*
-> idle_duration	*1*	*1*	*1*	*1*	*1*	*1*	*100.0%*	*50*

Fig. 7. Nova-G Compute performance over 50 VM boot and delete iterations with Rally.

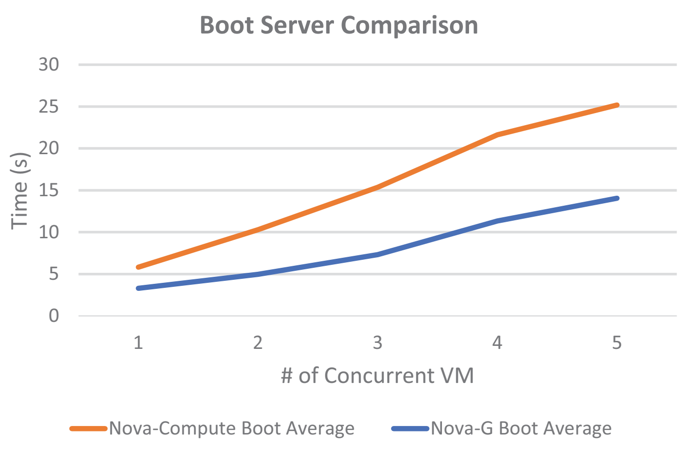


Fig. 8. Nova Compute and Nova-G Compute VM boot times.

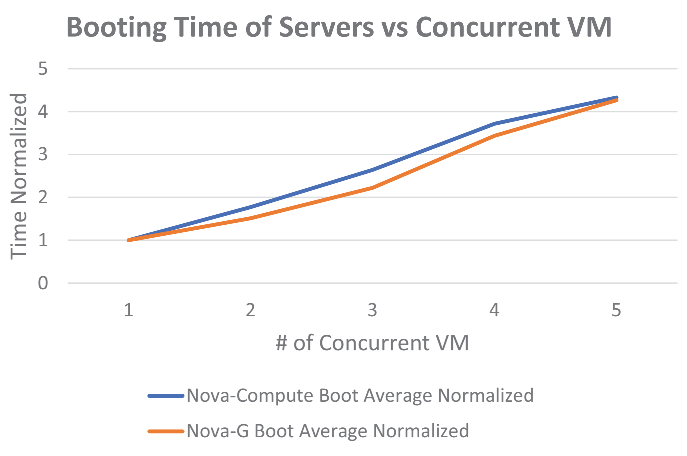


Fig. 9. Nova Compute and Nova-G Compute scalability.

For each test, we repeat 50 VM instantiations. Our sample means are within the respective percentage error of the true mean with a confidence level of 95% as shown in Table III.

TABLE III
CONFIDENCE LEVELS FOR THE TEST RESULTS.

Concurrent VMs	**1**	**2**	**3**	**4**	**5**
Error (%),	6.96	5.23	6.50	4.95	4.77

V. Conclusion and Future Work

In this paper, we propose a generalization for OpenStack resource allocation project Nova to accommodate the new types of resources in hardware accelerated clouds. To this end, we extend the database structures of OpenStack by the new types of resources such as FPGA or GPU. This extension is at the same level with the standard resources of OpenStack database which enables standard Nova Scheduler algorithms to work with these new resource types. More importantly we develop a new lightweight Nova Compute module that we call Nova-G Compute that works seamlessly with the standard OpenStack services and can work on any hardware platform that supports Rabbit MQ thanks to its implementation that does not have OS dependencies. We further develop an FPGAvisor that gives access to FPGA resources to Nova-G Compute through its generalized hypervisor driver.

Our experimental evaluations with the standard OpenStack benchmarking tool Rally show that Nova-G correctly communicates with other OpenStack components and can boot VMs on generalized resources without any performance degradation with respect to standard Nova Compute. In this paper, Nova-G Compute performance and functionality is evaluated on a test setup with virtual machines. Currently we are implementing Nova-G Compute for managing a real accelerated cloud system in a laboratory environment [29]. The FPGA accelerators are implemented on Zynq SoC platform.

Acknowledgment

The authors would like to thank TUBITAK and ASELSAN for their support.

References

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

[2] P. Cooke, J. Fowers, G. Brown, and G. Stitt, "A tradeoff analysis of fpgas, gpus, and multicores for sliding-window applications," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 1, p. 2, 2015.

[3] "Openstack global market revenues from 2014 and 2021," accessed: 2019-03-21. [Online]. Available: https://www.statista.com/statistics/498552/openstack-market-size/

[4] "What is openstack?" accessed: 2019-02-02. [Online]. Available: https://www.openstack.org/software/

[5] "Openstack docs: Compute api," accessed: 2019-02-02. [Online]. Available: https://developer.openstack.org/api-ref/compute/

[6] "Openstack docs: Keystone, the openstack identity service," accessed: 2019-02-02. [Online]. Available: https://docs.openstack.org/keystone/latest/

[7] "Openstack docs: Welcome to neutron's documentation!" accessed: 2019-02-02. [Online]. Available: https://docs.openstack.org/neutron/latest/

[8] "Openstack docs: Networking api v2.0," accessed: 2019-02-02. [Online]. Available: https://developer.openstack.org/api-ref/network/v2/index.html

[9] "Openstack docs: Welcome to glance's documentation!" accessed: 2019-02-02. [Online]. Available: https://docs.openstack.org/glance/latest/

[10] "Openstack docs: Horizon: The openstack dashboard project," accessed: 2019-02-02. [Online]. Available: https://docs.openstack.org/horizon/latest/

[11] S. Lima, Á. Rocha, and L. Roque, "An overview of openstack architecture: a message queuing services node," *Cluster Computing*, pp. 1–12, 2017.

[12] "Rabbitmq best practices," accessed: 2019-03-22. [Online]. Available: https://www.rabbitmq.com/

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

[14] "Amazon ec2 f1 instances," accessed: 2019-03-22. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/

[15] "Graphics processing unit (gpu) — google cloud," accessed: 2019-03-22. [Online]. Available: https://cloud.google.com/gpu/

[16] "Cloud tpus - ml accelerators for tensorflow — cloud tpu — google cloud," accessed: 2019-03-22. [Online]. Available: https://cloud.google.com/tpu/

[17] "Cloud iot core — cloud iot core — google cloud," accessed: 2019-03-22. [Online]. Available: https://cloud.google.com/iot-core/

[18] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim *et al.*, "A cloud-scale acceleration architecture," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 2016, p. 7.

[19] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "Fpgas in the cloud: Booting virtualized hardware accelerators with openstack," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2014, pp. 109–116.

[20] 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.

[21] "Xilinx zynq-7000 soc," accessed: 2019-03-22. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[22] 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.

[23] 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.

[24] 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.

[25] "Cyborg-nova-glance interaction in compute node," https://docs.openstack.org/cyborg/latest/specs/rocky/approved/compute-node.html, accessed: 2019-03-21.

[26] 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.

[27] "what is rally?" accessed: 2019-03-22. [Online]. Available: https://docs.openstack.org/developer/rally/

[28] "Overview rally," accessed: 2019-03-17. [Online]. Available: https://rally.readthedocs.io/en/latest/overview/overview.html

[29] "A novel fpga accelerated cloud architecture," accessed: 2019-03-22. [Online]. Available: http://accloud.eee.metu.edu.tr/