Test report for OpenStack cascading solution To support 1 million VMs In 100 data centers

1 Conclusion

According to the Phase I and Phase II test data analysis, due to the hardware resources limitation, the OpenStack cascading solution with current configuration can supports a maximum of 1 million virtual machines and is capable of handling 500 concurrent API request if L3 (DVR) mode is included or, 1000 concurrent API request if only L2 networking needed

It's up to deployment policy to use OpenStack cascading solution inside one site (one data center) or multi-sites (multi-data centers), the maximal sites (data centers) supported are 100, i.e., 100 cascaded OpenStack instances.

There is already report for one OpenStack instance to support 1k physical hosts, and it's also tested inside Huawei lab. The test report also shows that OpenStack cascading solution can manage up to 100k physical hosts without challenge.

If more hardware resources and more powerful hardware resources added to the test bed, higher capacity (more than 1 million virtual machines and ports and volumes) and higher API request concurrency can be achieved.

But if more features like security group, FWaaS is developed and included in the test, the API request concurrency will be degraded.

2 Before you read the report

2.1 Background

The background of OpenStack cascading solution is required.

The OpenStack cascading solution allows OpenStack to orchestrate multiple OpenStack instances located in one site, or at different geographic sites, or even from different vendors.

In this report, the test case is to use one cascading OpenStack to orchestrate 100 simulated cascaded OpenStack instances, to verify 1 million virtual machine scalability of OpenStack cascading solution. Those 100 simulated cascaded OpenStack instances can be located in one site (one data center), or 100 sites (100 data centers), it up to the deployment policy, because from technical point of view, the cascading OpenStack and the cascaded OpenStack communicate through standard OpenStack Restful API.

If you are the first time to learn about OpenStack cascading solution, it's recommended that you at least read the wiki page before you go through the report. https://wiki.openstack.org/wiki/OpenStack_cascading_solution

2.2 How to test

1. It takes many months to prepare the test bed: develop simulator, setup the test bed, and write the load generator script (not using Rally).The source code (https://github.com/stackforge/tricircle/) used in the performance test for OpenStack cascading solution is just for PoC only, not production ready source code.
2. The cascaded OpenStack simulator is developed based on single real OpenStack under load test with deviation correction.
3. One million virtual machines were created in the system step by step. Because we have to fix some bugs or adjust the environment configuration, or add more resources, or develop new simulated feature.
4. The test is divided into 2 phases.
5. In phase I, only L2 networks (VxLAN) were created in order to add virtual machine into the system. And only 100 projects were created, and there are 10k VMs per project. It's a mode with large size project. After the system successfully reach 1 million virtual machines, the API request concurrency test was executed and system pressure in idle-state is observed. In phase I, all resources of a project was only allocated in one cascaded OpenStack.L2 population is enabled in the cascaded layer.
6. In phase II, mixed small size project, medium size project and large size project were tested. And some project's resources distributed across multiple cascaded OpenStack. The DVR(distributed virtual router) is enabled in the cascading layer to make the project's networks inter-connected through DVR in different cascaded OpenStack, and L2 population is also enabled for VxLAN network, but not test L2 network cross cascaded OpenStack. After 1million virtual machines were created successfully in the system, the concurrency test for router-add-interface, router-remove-interface was executed.
7. Single OpenStack to support 1k physical hosts and 10k virtual machines: There are cases reported that single OpenStack can be scaled to 1k hosts or10k virtual machine, for example: https://www.mirantis.com/blog/benchmarking-openstack-megascale-tested-mirantis-openstack-softlayer/, http://www.opencontrail.org/openstack-neutron-at-scale/

2.3 What's tested

1. Nova Cascading: VM boot, delete, start, stop, reboot, query
2. Cinder Cascading: Volume create, delete.
3. Neutron Cascading: L2 network (VxLAN), Router (DVR)
4. Glance cascading: Glance as the image location
5. Shared KeyStone: using PKI token

2.4 What's not tested

Lots of functions are not tested in this report. Especially functionality like security group, FWaaS, VPNaaS, LBaaS have not been developed yet in the

PoC source code of OpenStack cascading solution. On the other hand, the security group and FWaaS will produce performance impact on the scalability.

From theoretic, Neutron is like a “broadcast” domain, for example, enforcement of DVR and security group has to touch each regarding host where there is VM of the project resides. If there are plenty of physical hosts, for example, 10k, inside one Neutron, it’s very hard to overcome the “broadcast storm” issue, that’s the bottleneck for scalability. Just like the TCP/IP layered networking, the Ethernet L2 network is a broadcast domain, but the L3 can stop the broadcast storm and forward to another broadcast domain, that make the network can grow to scale. Through the Neutron cascading, the “broadcast storm” can also be solved by layered architecture, for example, just think about that fan-out one RPC message to 100 nodes in the cascading layer, and then forward the request to cascaded layer, and fan-out to 1000 nodes in each cascaded OpenStack, the broadcast storm is easy to solve in small number of nodes (100 or1000 compared to 100k) inside one OpenStack. Of course, the practical RPC interaction is different case by case, for example, DVR router-update will only lead to limited involved nodes on which there are VMs connecting to the DVR.

For “broadcast storm” like features, only DVR and L2 population has been tested, other features like security group, FWaaS, etc need to be developed and tested. From theoretic, it could be settled.

2.5 Numbers

1. In phase I, all load tests have been done under the system already populated with 1 million VMs. All VMs are populated through the cascading OpenStack API.

```
MariaDB [nova]> select vm_state,count(*) from instances group by vm_state;
+----------+----------+
| vm_state | count(*) |
+----------+----------+
| active   |  1001999 |
| building |     4555 |
+----------+----------+
2 rows in set (1.25 sec)
```

   Each VMs has one port, but unfortunately the screen of ports number has not been captured in phase I.

2. In phase II, all load tests have been done also under the system already populated with 1 million VMs. All VMs are populated through the cascading OpenStack API.

   Because the lab was powered off suddenly, the database in the cascading layer was damaged, and the Cinder database cannot be recovered, it’s little pity that no backup policy has been applied for the database.

   During the recover stage, new VM creation report no IP address, and the tester manipulate the Neutron DB directly, and unexpectedly remove IP address in the IP table, and make some ports (about 200k) lost IP address, and hard to recover the IP address to the port manually from the cascaded OpenStack instances, at last these orphan ports have to be removed from the cascading layer. After the operation, the lab planned to be powered off again only 1 day after that.

```
MariaDB [nova]> select vm_state,count(*) from instances group by vm_state;
+----------+----------+
| vm_state | count(*) |
+----------+----------+
| active   |  1122053 |
| deleted  |       10 |
| error    |        1 |
+----------+----------+
3 rows in set (1.59 sec)
```

```
MariaDB [neutron]> select status,count(*) from ports group by status;
+--------+----------+
| status | count(*) |
+--------+----------+
| ACTIVE |   916595 |
| DOWN   |    11969 |
+--------+----------+
2 rows in set (2.07 sec)
```

We have to release the resources for other test purpose and at last no chance to recover the system to previous health system. The figures captured here is what we can presented in the report.

3. Everyone is encouraged establish test environment to verify the scalability of OpenStack cascading proposal.

2.6 CPU usage

If you find a CPU usage is greater than 100%, then it means that it's the summary CPU usage for all same service's processes running on different hyper thread or vCPU.

3 Test Bed

3.1 Hardware Infrastructure

Figure 3-1 Physical Networking

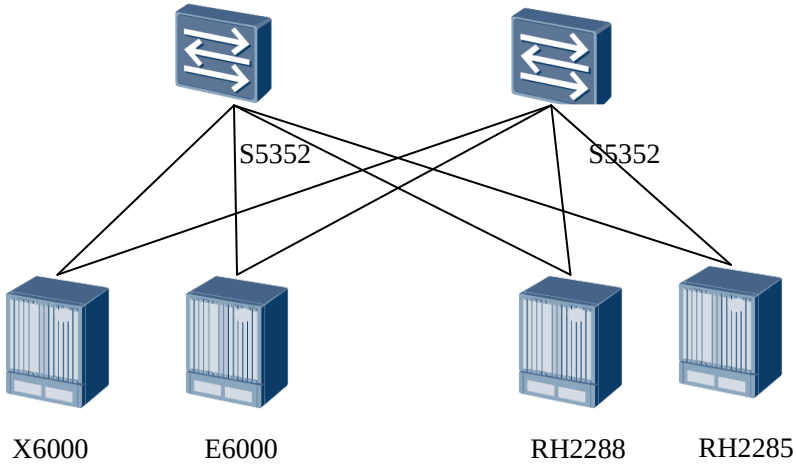


Table 3-1 Configuration

Resource Type	Model	Description
Compute	Huawei E6000 (27 servers) Huawei X6000 (5 servers) Huawei RH2285 (1 server) Huawei RH2288 (12 servers)	The system consists of 45 servers

Resource Type	Model	Description
Network	Broadcom Corporation NetXtreme BCM5715S Gigabit Ethernet	Controller nodes and compute nodes use 1GE common network interface cards (NICs).
Storage (local disks on servers)	SAS (40 disks) SATA	Database servers use SAS disks. Other services use SATA disks. All disks are configured in 10-disk RAID 10 and provide LVM storage.

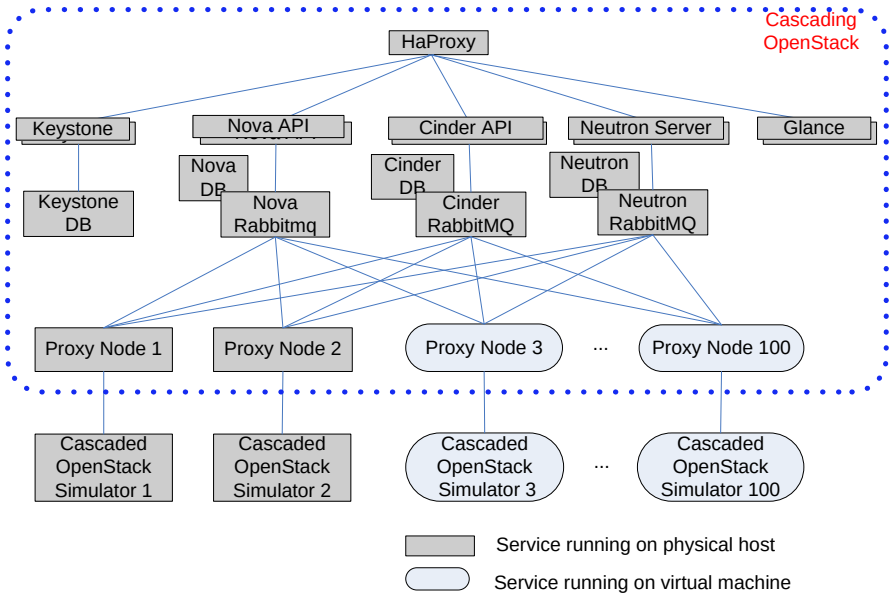
3.2 Software Packages

Table 3-2 Software Packages

Software	Version
Cascading OpenStack	OpenStack Juno with cascading patches from https://github.com/stackforge/tricircle
Cascaded OpenStack	OpenStack Juno with cascading patches from https://github.com/stackforge/tricircle
Operating system (OS)	Ubuntu 14.04 LTS
Database	5.5.39-MariaDB
RabbitMQ	3.2.4

3.3 Software deployment

Figure 3-2 Software Deployment



3.4 The Cascading OpenStack

Cascading controller nodes are deployed on physical hosts, and proxy node (two of them running on physical hosts) is deployed on virtual machine, thereby cascading OpenStack is a real system.

3.5 The Cascaded OpenStack Simulator

Cascaded OpenStack is deployed in All-in-One mode or simulator mode.

Cascaded OpenStack simulator is developed based on the real OpenStack behavior under load.

Cascaded OpenStack simulator is able to receive requests from cascading OpenStack and respond to resource allocation requests, thereby simulating close-to-real workloads. Cascaded OpenStack simulator is developed based on real OpenStack's

The cascaded OpenStack simulators use stubs on nova-api, cinder-api, and neutron-api to create API responses, instead of issuing the API requests to underlying modules.

These simulators are able to respond to all requests from the cascading layer and return corresponding data (for example, data about instances, images, and networks). Therefore, the cascading layer recognizes the simulators as real cascaded OpenStack systems.

Stubs on the API layer allow the simulators to run only API-related services, thereby minimizing resource consumption of the simulators. In the test, each simulator is deployed on an virtual machine created in the Huawei FusionCompute hypervisor (CNA node) to simulate a cascaded OpenStack system.

3.6 Test Tools

Table 3-3 Test tools

Tool	Description
SoapUI	Sends REST messages.
Shell script	Issues messages and implements automatic system configuration.

Use SoapUI and shell scripts to issue requests concurrently, consistently to create instances, thereby increasing loads on the system. Testing the maximum number of instances the cascading system can support and evaluate the system performance.

3.7 Basic Configuration

Item	Quantity	Remarks
Cascading OpenStack	1	One cascading OpenStack including service KeyStone, Glance, Nova, Cinder and Neutron. Ceilometer is not included.

Item	Quantity	Remarks
Proxy Node	100	One proxy node which will run Nova-proxy, Cinder-Proxy, Neutron Proxy to manage one cascaded OpenStack
Cascaded OpenStack	100	
Availability zone (AZ)	100	one cascaded OpenStack configured with one AZ.

3.8 Software Installation and configuration

Role	Server				Deployment Mode
	Model	CPU (x 2)	Memory	Hard disk	
Load generator	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple nodes are deployed on a host in sharing mode.
HAProxy	RH2285	Intel(R) Xeon(R) CPU E5620 @ 2.40 GHz	48 GB	A RAID 10 array with 10 SATA member disks	Exclusively deployed on a phiscial host.
Keystone1	RH2288	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz	80 GB	A RAID 10 array with 10 SATA member disks	Exclusively deployed on a phiscial host.
Keystone 2	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple nodes are deployed on a host in sharing mode
Glance	RH2288	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz	48 GB	A RAID 10 array with 10 SATA member disks	Exclusively deployed on a phiscial host.
Nova API 1	RH2288	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz	48 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.
Nova API 2	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.
Nova API 3	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.
Nova API 4	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.
Nova API 5	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.

Role	Server				Deployment Mode
Nova scheduler 1	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.
Nova scheduler 2	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.
Nova scheduler 3	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.
RabbitMQ (Nova)	X6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	144 GB	A RAID 10 array with 10 SATA member disks	Exclusively deployed on a phiscial host.
Nova DB	RH2288	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz	48 GB	A RAID 10 array with 10 SAS member disks	Exclusively deployed on a phiscial host.
Cinder API	RH2288	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz	48 GB	A RAID 10 array with 10 SATA member disks	Exclusively deployed on a phiscial host.
RabbitMQ (Cinder)	X6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	144 GB	A RAID 10 array with 10 SATA member disks	Exclusively deployed on a phiscial host.
Cinder DB	RH2288	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz	48 GB	A RAID 10 array with 10 SAS member disks	Exclusively deployed on a phiscial host.
Neutron server 1	RH2288	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz	64 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.
Neutron server 2	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.
Neutron server 3	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.
Neutron server 4	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.

Role	Server				Deployment Mode
Neutron server 5	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts.
Neutron server 6	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Distributively deployed on multiple phiscial hosts..
RabbitMQ (Neutron)	X6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	144 GB	A RAID 10 array with 10 SATA member disks	Exclusively deployed on a phiscial host.
Neutron DB	RH2288	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz	48 GB	A RAID 10 array with 10 SAS member disks	Exclusively deployed on a phiscial host.
Proxy Node 1	RH2288	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz	48 GB	A RAID 10 array with 10 SATA member disks	Exclusively deployed on a phiscial host.
Proxy Node 2	RH2288	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz	64 GB	A RAID 10 array with 10 SATA member disks	Exclusively deployed on a phiscial host.
CNA 1 (Proxy Nodes)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple proxy node VMs are deployed on a host in sharing mode.
CNA 2 (Proxy Nodes)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple proxy node VMs are deployed on a host in sharing mode.e.
CNA 3 (Proxy Nodes)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple proxy node VMs are deployed on a host in sharing mode.
CNA 4 (Proxy Nodes)	X6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple proxy node VMs are deployed on a host in sharing mode..
CNA 5 (Proxy Nodes)	X6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	40 GB	A RAID 10 array with 10 SATA member disks	Multiple proxy node VMs are deployed on a host in sharing mode.
CNA 6 (Proxy Nodes)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple proxy node VMs are deployed on a host in sharing mode.

Role	Server				Deployment Mode
Cascaded OpenStack simulator 1	RH2288	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz	64 GB	A RAID 10 array with 10 SATA member disks	Exclusively deployed on a host in All-in-One mode.
Cascaded OpenStack simulator 2	RH2288	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10 GHz	64 GB	A RAID 10 array with 10 SATA member disks	Exclusively deployed on a host in All-in-One mode.
CNA 1 (cascaded OpenStack simulator)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple VMs are deployed on a host in sharing mode.
CNA 2 (cascaded OpenStack simulator)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple VMs are deployed on a host in sharing mode.
CNA 3(cascaded OpenStack simulator)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple VMs are deployed on a host in sharing mode.
CNA 4 (cascaded OpenStack simulator)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple VMs are deployed on a host in sharing mode.
CNA 5 (cascaded OpenStack simulator)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple VMs are deployed on a host in sharing mode.
CNA 6 (cascaded OpenStack simulator)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	48 GB	A RAID 10 array with 10 SATA member disks	Multiple VMs are deployed on a host in sharing mode.
CNA 7 (cascaded OpenStack simulator)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	80 GB	A RAID 10 array with 10 SATA member disks	Multiple VMs are deployed on a host in sharing mode.
CNA 8 (cascaded OpenStack simulator)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	80 GB	A RAID 10 array with 10 SATA member disks	Multiple VMs are deployed on a host in sharing mode.
CNA 9 (cascaded OpenStack simulator)	E6000	Intel(R) Xeon(R) CPU E5645 @ 2.40 GHz	96 GB	A RAID 10 array with 10 SATA member disks	Multiple VMs are deployed on a host in sharing mode.

4 Phase I Test including L2 for the cascading OpenStack under the semi-simulated test bed

4.1 Performance analysis when system in concurrent API request

4.1.1 Test Content

This test is conducted for one cascading OpenStack with 100 cascaded OpenStack instances accommodating one million instances to verify the service concurrency performance. The test focuses on the processing capabilities and resource consumption of all services at the cascading layer under load test. Table 4-1 lists the test content.

Table 4-1 Service concurrency test content

Category	Test Content	Concurrent API Request	Remarks
Instance life cycle management	Concurrently creating instances	100/200/500/1000	100 tenants can concurrently perform operations.
	Concurrently deleting instances	100/200/500/1000	
	Concurrently starting instances	100/200/500/1000	
	Concurrently stopping instances	100/200/500/1000	
	Concurrently restarting instances	100/200/500/1000	
	Concurrently querying instance details	100/200/500/1000	
Volume management	Concurrently creating volumes	100/200/500/1000	
	Concurrently deleting volumes	100/200/500/1000	
	Concurrently querying volume details	100/200/500/1000	
Network management	Concurrently creating networks	100/200/500/1000	
	Concurrently deleting networks	100/200/500/1000	
	Concurrently querying network details	100/200/500/1000	

Category	Test Content	Concurrent API Request	Remarks
	Concurrently creating subnets	100/200/500/1000	
	Concurrently deleting subnets	100/200/500/1000	
	Concurrently querying subnet details	100/200/500/1000	

4.1.2 Test Result

Nova

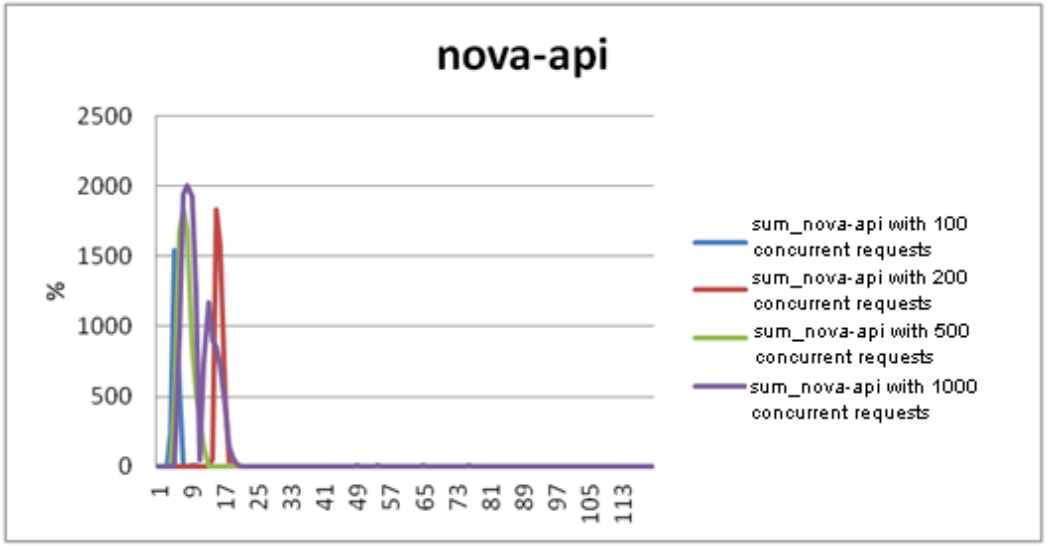
nova-api

The nova-api processes and the nova-conductor processes are co-located on five servers, and each nova-api process is equipped with eight workers. The resource usage data provided in the following is collected for all nova-api processes.

When instances are concurrently created, deleted, started, and stopped, the CPU usage of the nova-api processes reaches over 2000%, and the duration (in seconds) depends on the number of concurrent requests. After the concurrent requests are processed, the CPU usage falls back.

Figure 4-1shows the CPU usages of the nova-api processes when instances are concurrently created.

Figure 4-1 CPU usages of the nova-api processes during the concurrent instance creation



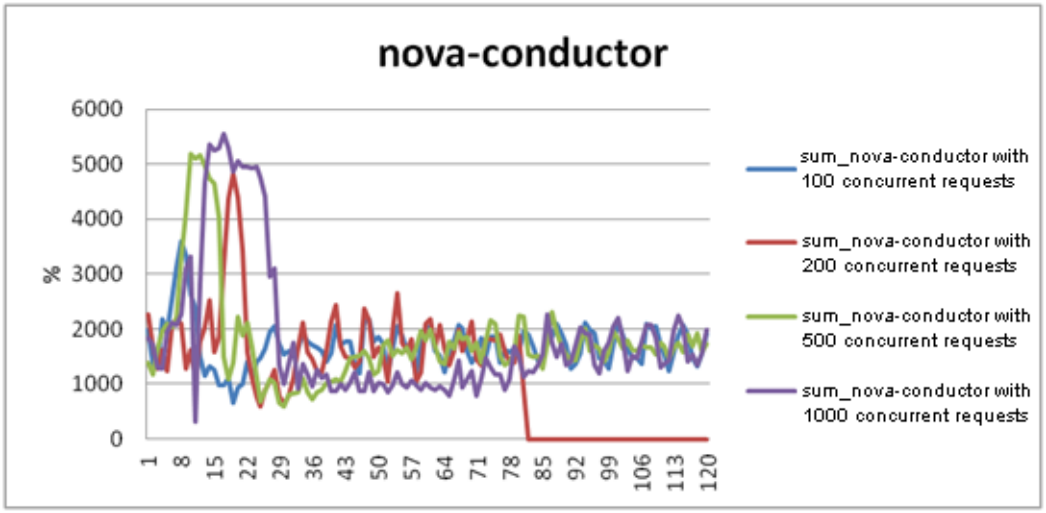
nova-conductor

The nova-conductor processes and the nova-api processes are co-located on five servers, and each nova-conductor process is equipped with 20 workers.

When instances are concurrently created, deleted, started, and stopped, the CPU usage of the nova-conductor processes reaches over 5000%, and the duration (in seconds) depends on the number of concurrent requests. After the concurrent requests are processed, the CPU usage falls back.

Figure 4-2 shows the CPU usages of the nova-conductor processes when instances are concurrently created.

Figure 4-2 CPU usages of the nova-conductor processes during the concurrent instance creation



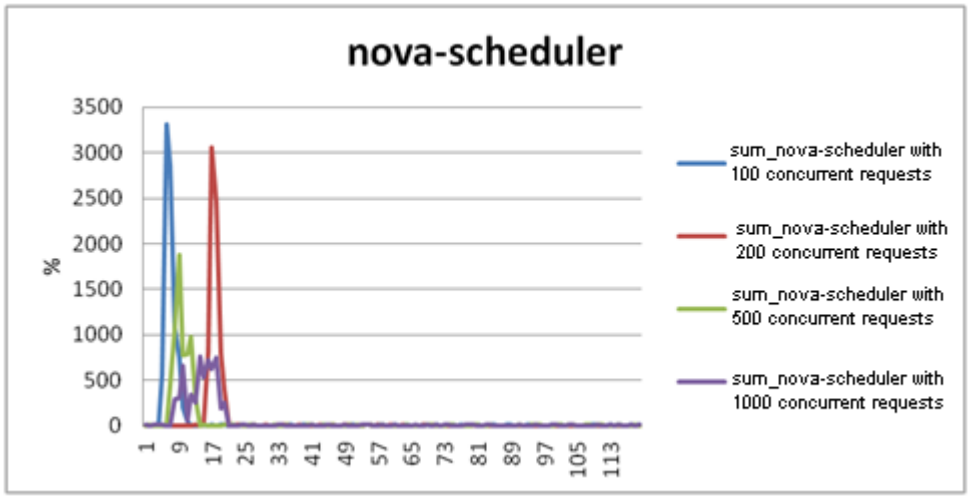
nova-scheduler

The nova-scheduler processes are deployed on three servers, and each nova-scheduler process is equipped with 18 workers.

When instances are concurrently created, deleted, started, and stopped, the CPU usage of the nova-scheduler processes reaches over 3000%, and the duration (in seconds) depends on the number of concurrent requests. After the concurrent requests are processed, the CPU usage falls back.

Figure 4-3 shows the CPU usages of the nova-scheduler processes when instances are concurrently created.

Figure 4-3 CPU usages of the nova-scheduler processes during the concurrent instance creation



(Nova-Sheduler does not support multi-worker feature in Juno version, it was added during the test. You have to put Nova-scheduler into more virtual machine or put more physical server to meet the need for service concurrency, otherwise, 3 physical servers is not enough.)

Cinder

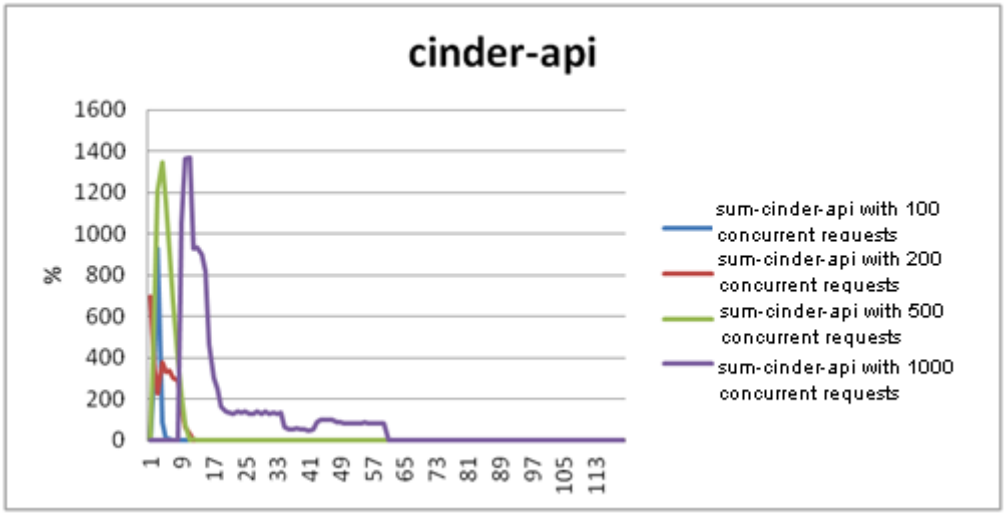
cinder-api

The cinder-api process and the cinder-scheduler process are co-located on an independent server.

The resource usage of the cinder-api processes fluctuates similarly when volumes are concurrently created, deleted, and queried.

Figure 4-4 shows CPU usages of the cinder-api processes when volumes are concurrently created.

Figure 4-4 CPU usages of the cinder-api processes during the concurrent volume creation



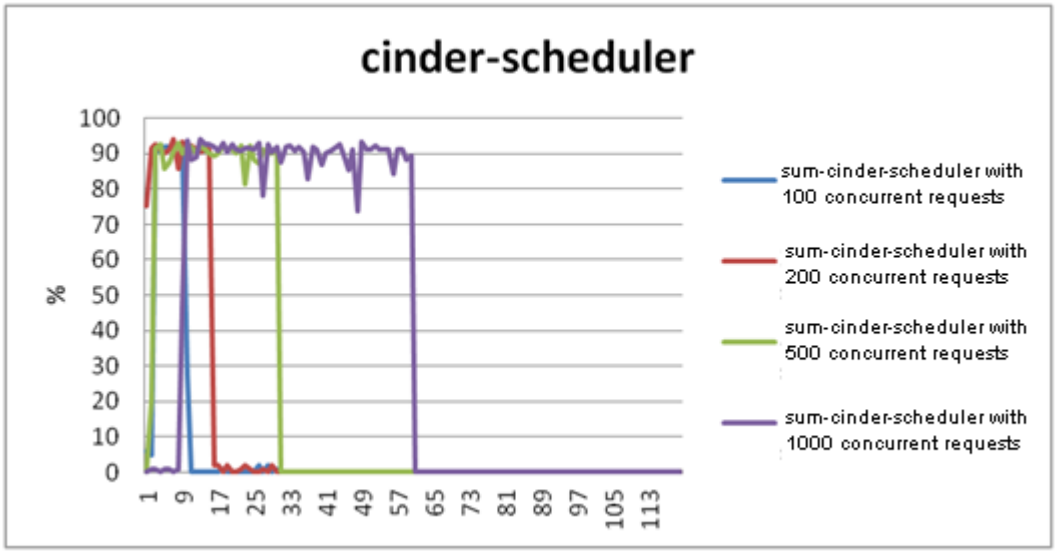
cinder-scheduler

The cinder-scheduler process and the cinder-api process are co-located on an independent server.

During the concurrent service processing period, the CPU usage of the cinder-scheduler process transiently reaches about 90%, but it becomes normal after the concurrent services are processed.

Figure 4-5 shows the CPU usages of the cinder-scheduler processes when volumes are concurrently created.

Figure 4-5 CPU usages of the cinder-scheduler processes during the concurrent volume creation



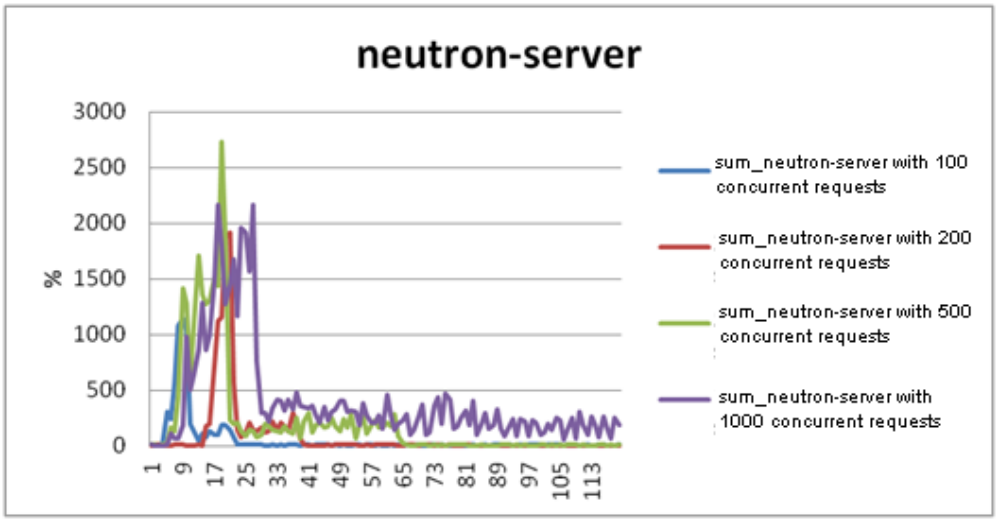
Neutron

The neutron-server processes are deployed on six servers, each of which is equipped with 12 RPC workers and 12 API workers.

The concurrent instance creation incurs a series of operations, including querying networks, subnets, and ports as well as creating ports. In this case, the CPU usage of the neutron-server process transiently reaches over 2000%.

Figure 4-6 shows the CPU usages of the neutron-server processes when instances are concurrently created.

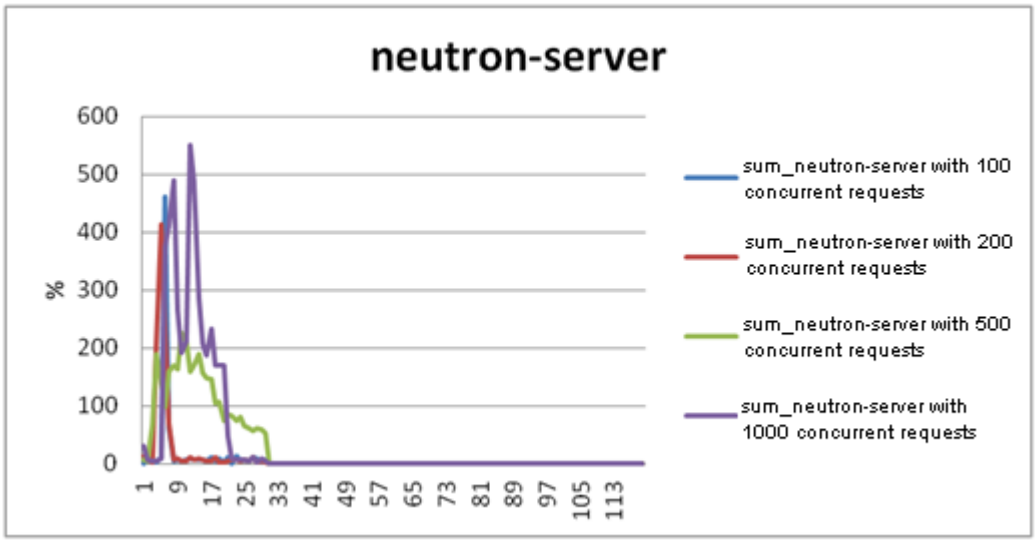
Figure 4-6 CPU usages of the neutron-server processes during the concurrent instance creation



The concurrent creation of networks and subnets is related only to the databases at the cascading layer, and therefore the system load is light.

Figure 4-7 shows the CPU usages of the neutron-server processes when networks are concurrently created.

Figure 4-7 CPU usages of the neutron-server processes during the concurrent network creation



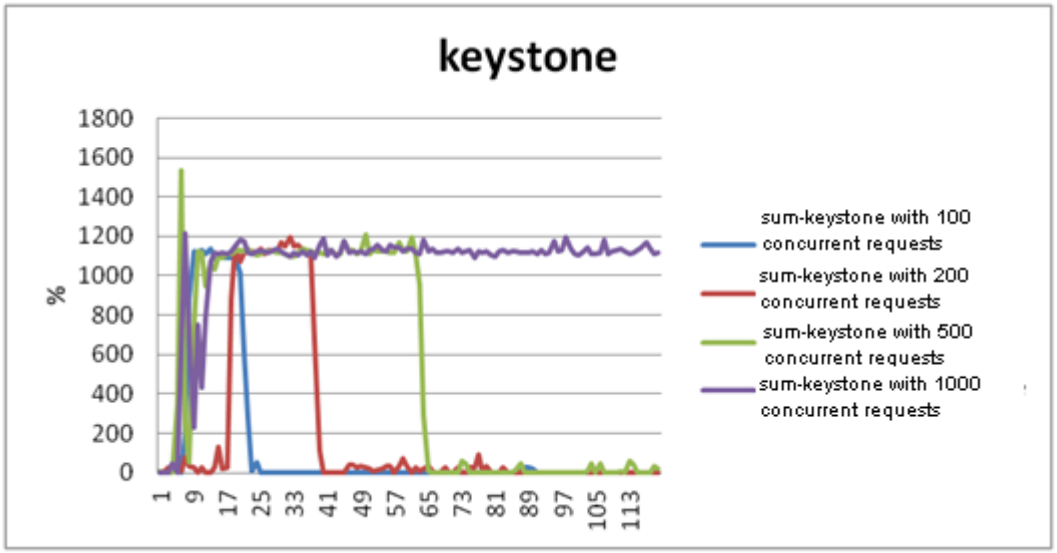
Keystone

The Keystone service is deployed on an independent server.

In the scenario with 100 concurrent tenants, the CPU usage of the Keystone processes reaches about 1200%, but it becomes normal after the concurrent requests are processed.

Figure 4-8 shows the CPU usages of the Keystone processes in the scenario with 100 concurrent tenants.

Figure 4-8 CPU usages of the Keystone processes in the scenario with 100 concurrent tenants



DB

Nova database

During the concurrent instance creation period, the CPU usage of the Nova database processes transiently reaches over 450%.

Figure 4-9 shows the CPU usages of the Nova database processes when instances are concurrently created.

Figure 4-9 CPU usages of the Nova database processes during the concurrent instance creation

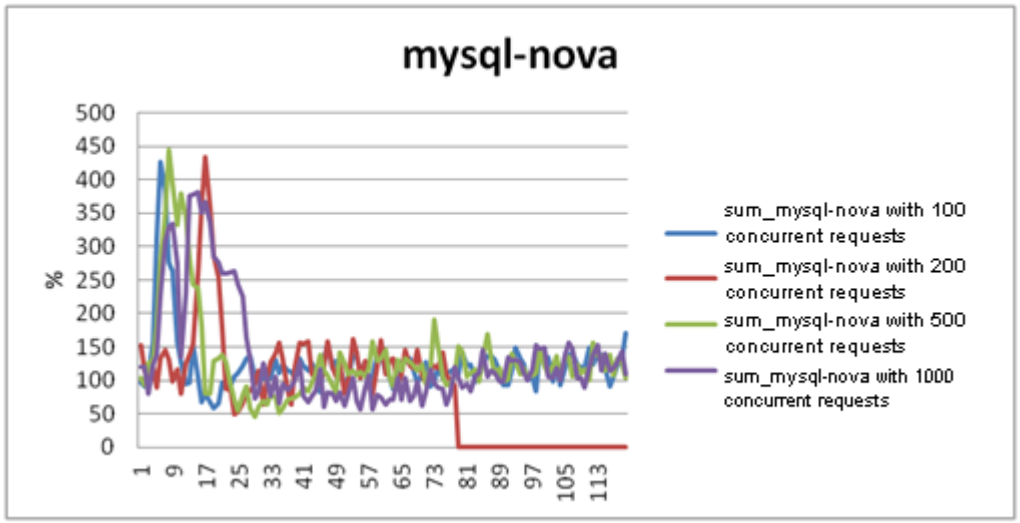


Figure 4-10 shows the IOPS distribution of the Nova database when 1000 instances are concurrently created.

Figure 4-10 IOPS distribution of the Nova database during the concurrent creation of 1000 instances

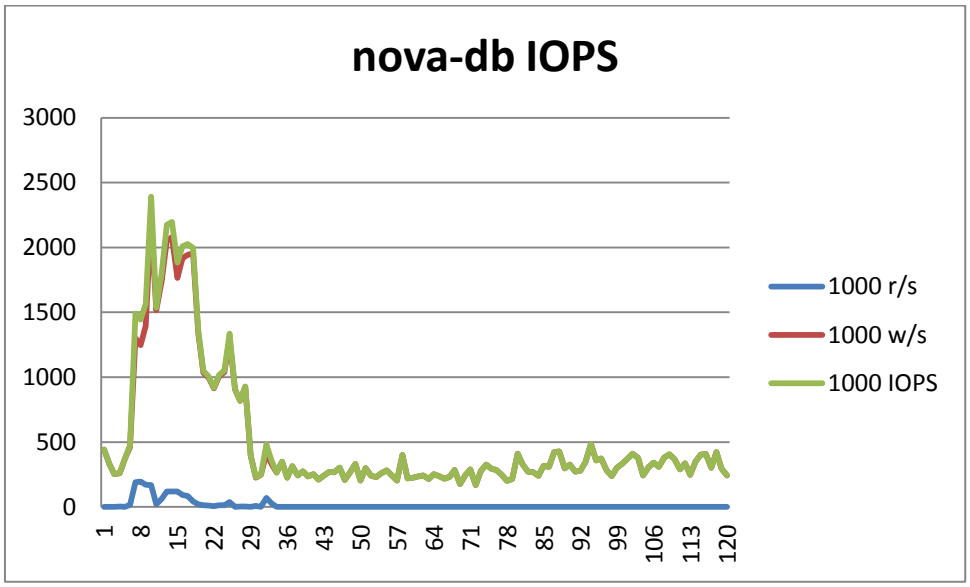
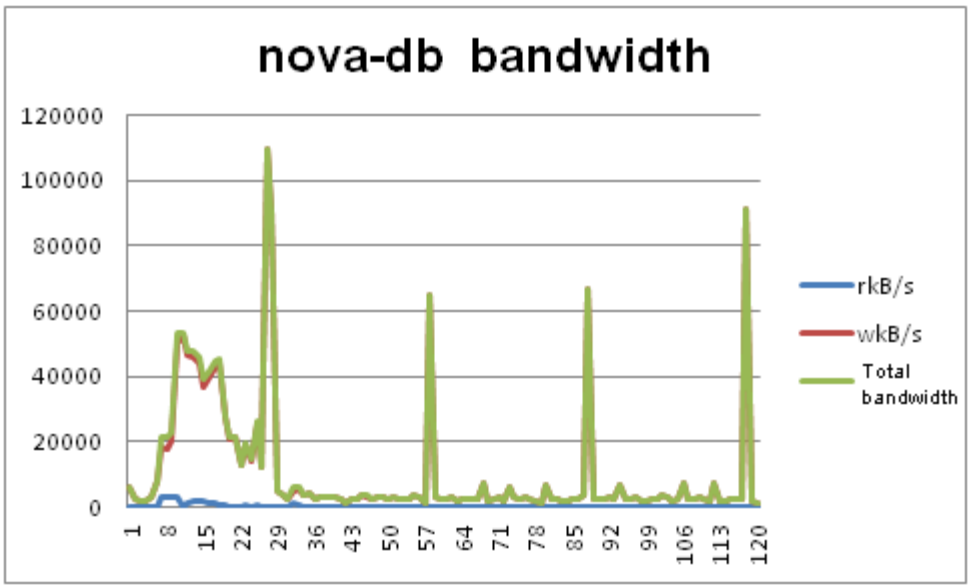


Figure 4-11 shows the storage bandwidth distribution of the Nova database when 1000 instances are concurrently created.

Figure 4-11 Storage bandwidth distribution of the Nova database during the concurrent creation of 1000 instances

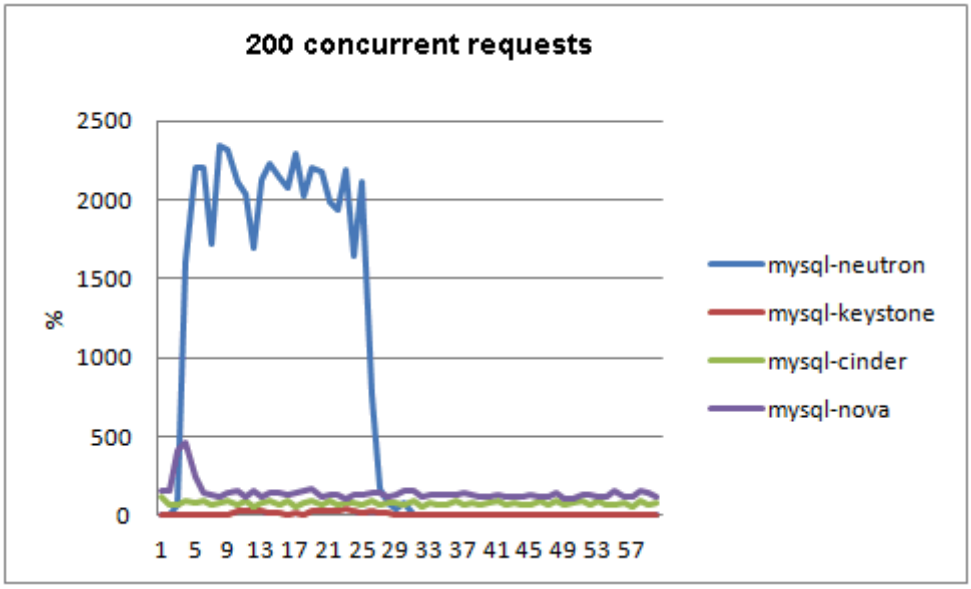


Neutron database

During the concurrent instance creation period, the CPU usage of the Neutron database reaches over 2000%.

Figure 4-12 shows the CPU usages of the Neutron database processes when instances are concurrently created before indexes are added.

Figure 4-12 CPU usages of the Neutron database processes during the concurrent instance creation before indexes are added



The abnormal CPU usage fluctuation is caused by the absence of key indexes for querying ports. The CPU usage becomes normal after indexes are added.

The ports table lacks key index for query using (tenant_id + device_id) or (network_id + mac_address) or (network_id + device_owner), it takes more than 1 second each query.

Figure 4-13 shows the CPU usages of the Neutron database processes when instances are concurrently created after indexes are added.

Figure 4-13 CPU usages of the Neutron database processes during the concurrent instance creation after indexes are added

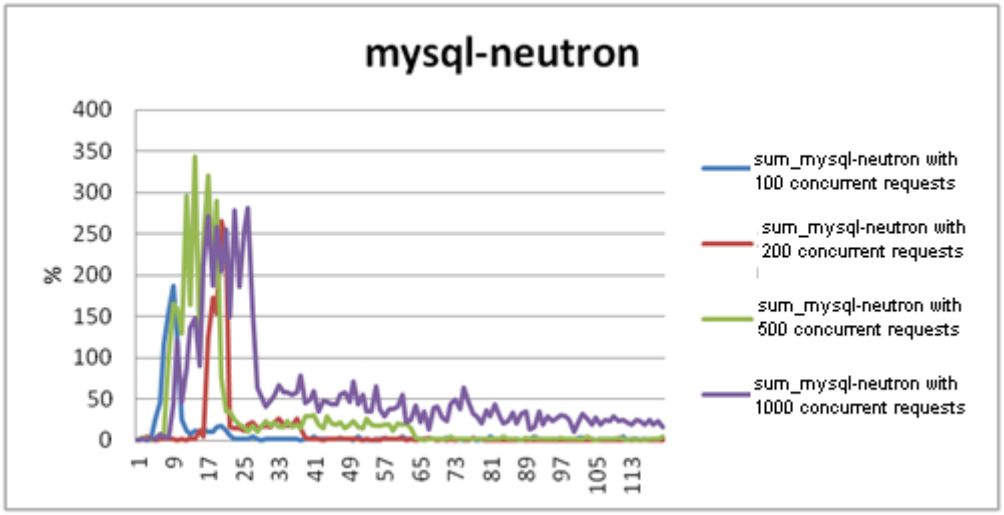
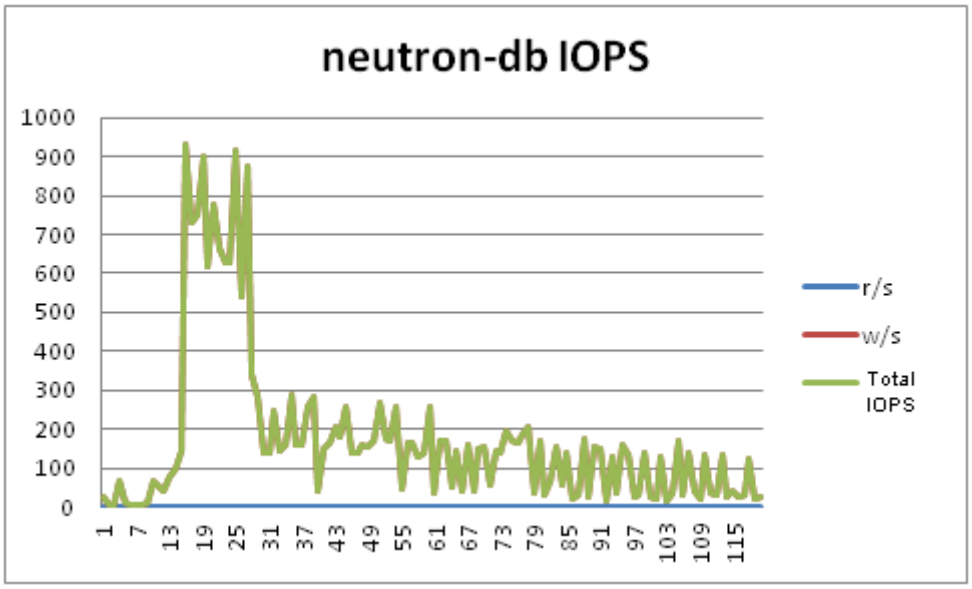


Figure 4-14 shows the IOPS distribution of the Neutron database when 1000 instances are concurrently created.

Figure 4-14 IOPS distribution of the Neutron database during the concurrent creation of 1000 instances



Keystone database

In the scenario with 100 concurrent tenants, the resource usage of the Keystone database fluctuates properly.

Figure 4-15 shows the CPU usages of the Keystone database in the scenario with 100 concurrent tenants.

Figure 4-15 CPU usages of the Keystone database in the scenario with 100 concurrent tenants

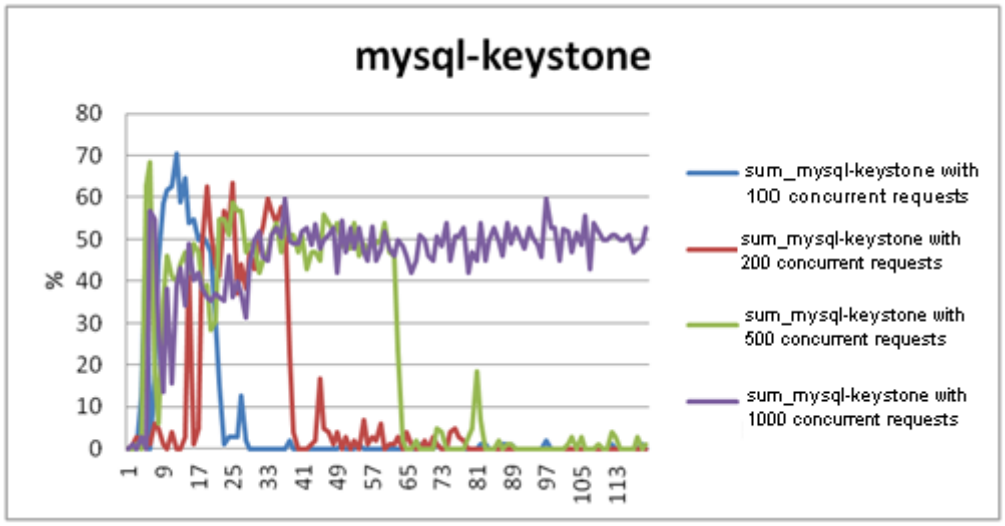


Figure 4-16 shows the IOPS distribution of the Keystone database when 1000 instances are concurrently created.

Figure 4-16 IOPS distribution of the Keystone database during the concurrent creation of 1000 instances

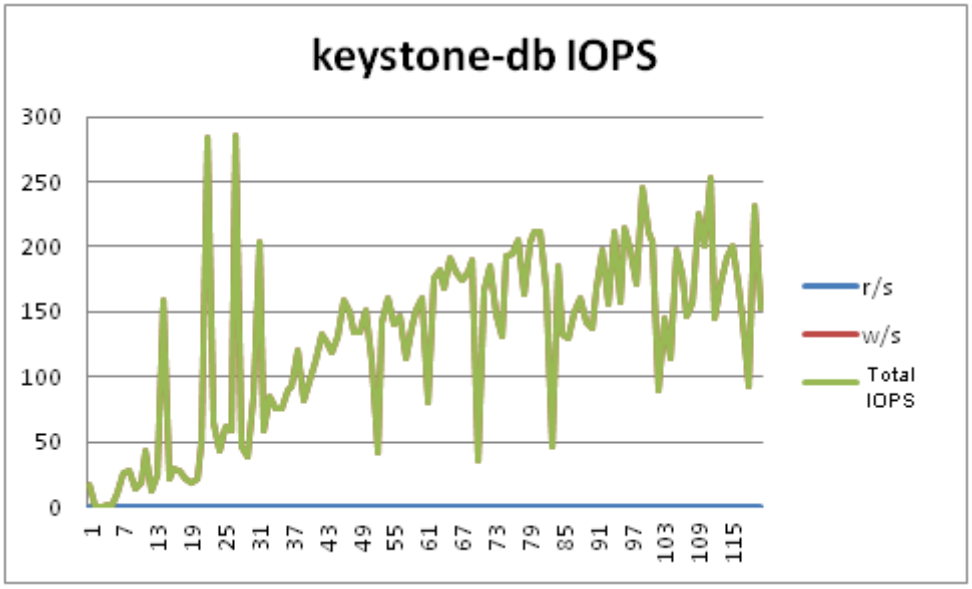
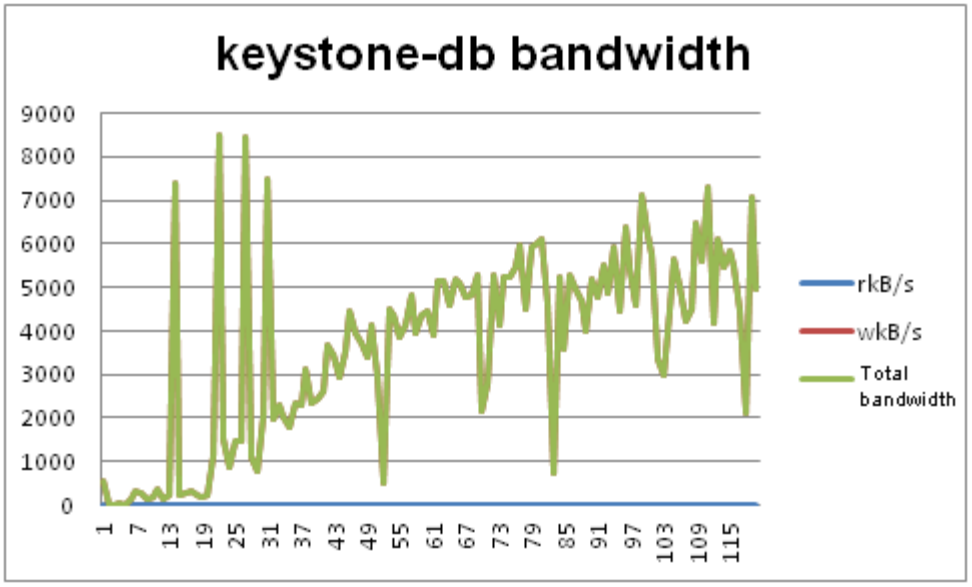


Figure 4-17 shows the storage bandwidth distribution of the Keystone database when 1000 instances are concurrently created.

Figure 4-17 Storage bandwidth distribution of the Keystone database during the concurrent creation of 1000 instances



Cinder database

When operations, such as creating, querying, and deleting, are concurrently performed for volumes, the CPU usage of the Cinder database transiently reaches about 200%.

Figure 4-18 shows the CPU usages of the Cinder database when volumes are concurrently created.

Figure 4-18 CPU usages of the Cinder database during the concurrent volume creation

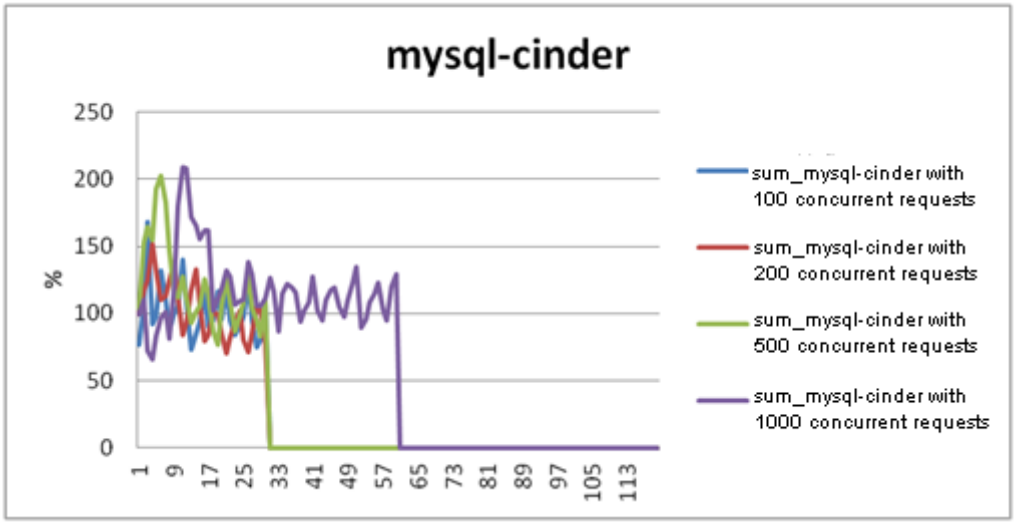


Figure 4-19 shows the IOPS distribution of the Cinder database when volumes are concurrently created.

Figure 4-19 IOPS distribution of the Cinder database during the concurrent creation of 1000 volumes

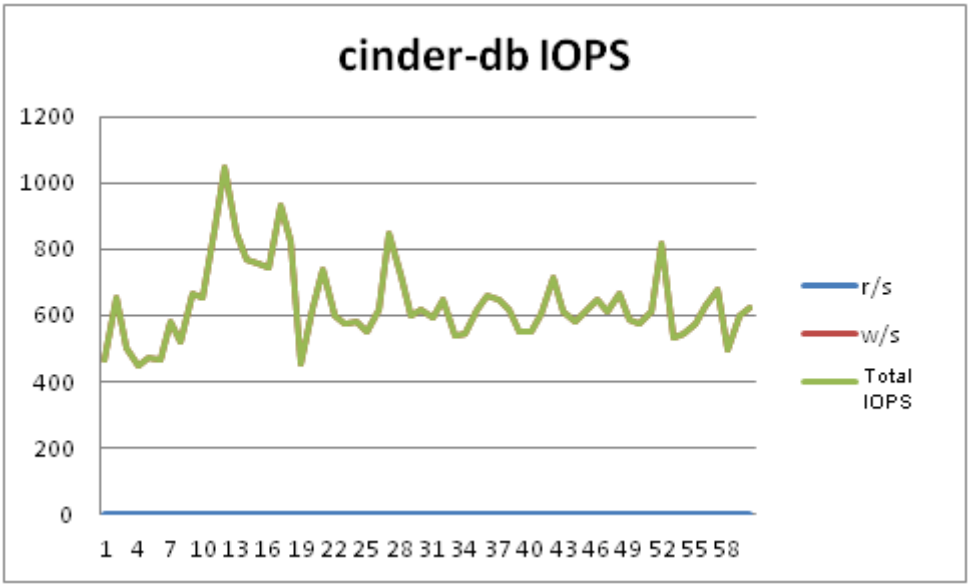
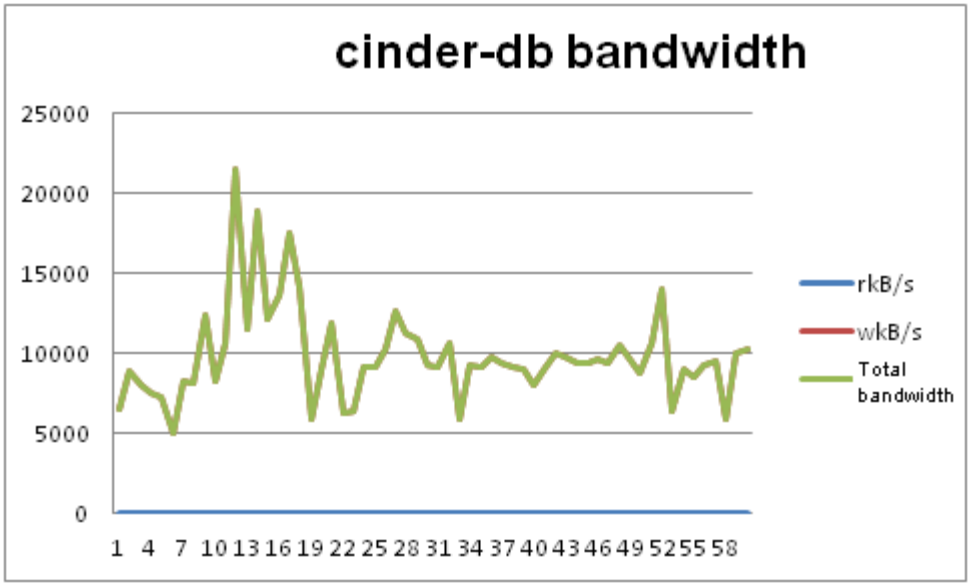


Figure 4-20 shows the storage bandwidth distribution of the Cinder database when volumes are concurrently created.

Figure 4-20 Storage bandwidth distribution of the Cinder database during the concurrent creation of 1000 volumes



RabbitMQ

Nova

When instances are concurrently created, deleted, started, stopped, or queried, the CPU usage of the Nova RabbitMQ processes reaches about 650% periodically.

Figure 4-21 shows the CPU usages of the Nova RabbitMQ processes when instances are concurrently created.

Figure 4-21 CPU usages of the Nova RabbitMQ processes during the concurrent instance creation

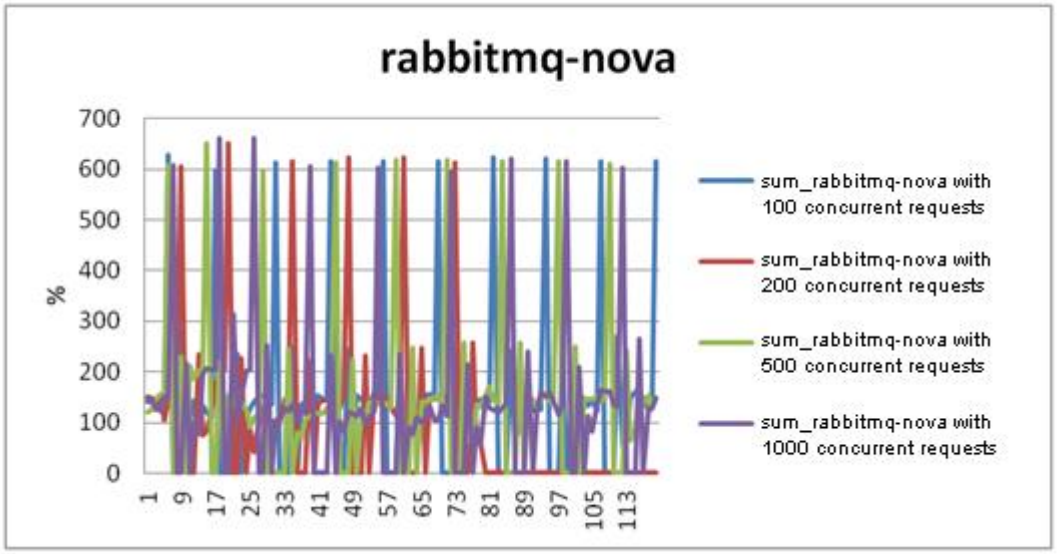
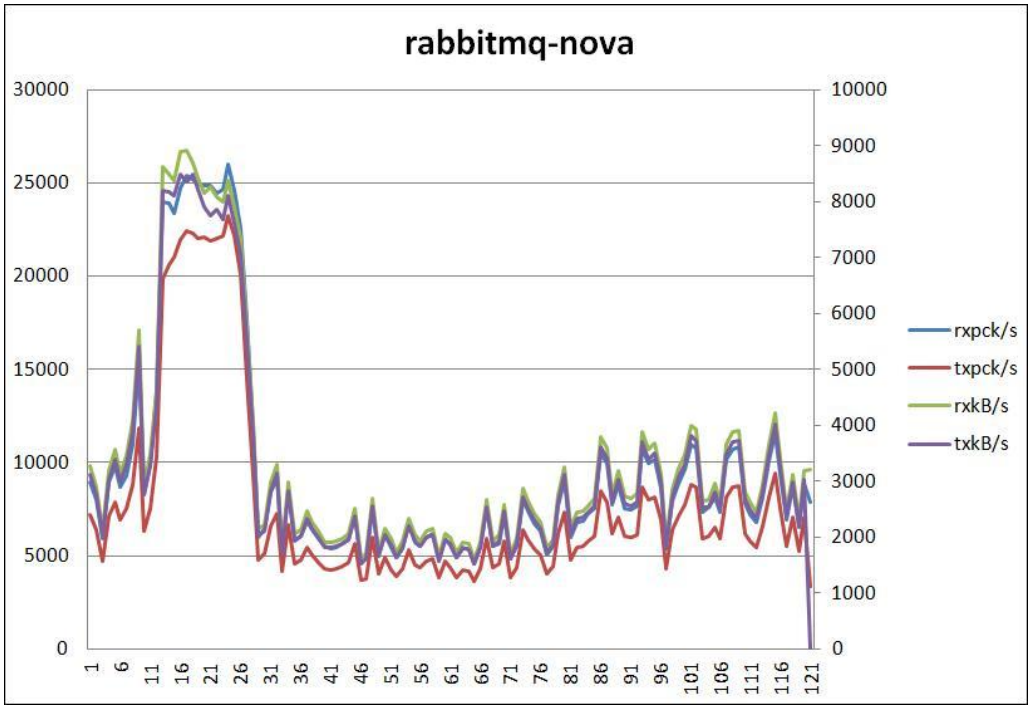


Figure 4-22 shows the network loads of the Nova RabbitMQ node when instances are concurrently created.

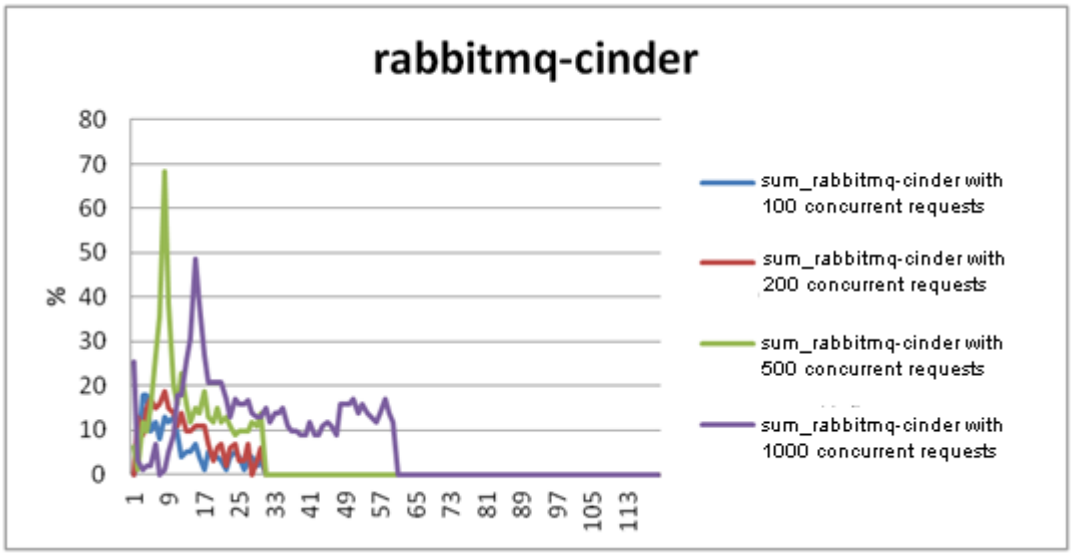
Figure 4-22 Network loads of the Nova RabbitMQ node during the concurrent instance creation



Cinder

Figure 4-23 shows the CPU usages of the Cinder RabbitMQ processes when instances are concurrently created.

Figure 4-23 CPU usages of the Cinder RabbitMQ processes during the concurrent instance creation



Neutron

When instances are concurrently created, the CPU usage of the Neutron RabbitMQ processes transiently reaches about 500% and falls back after the creation task is completed.

Figure 4-24 shows the CPU usages of the Neutron RabbitMQ processes when instances are concurrently created.

Figure 4-24 CPU usages of the Neutron RabbitMQ processes during the concurrent instance creation

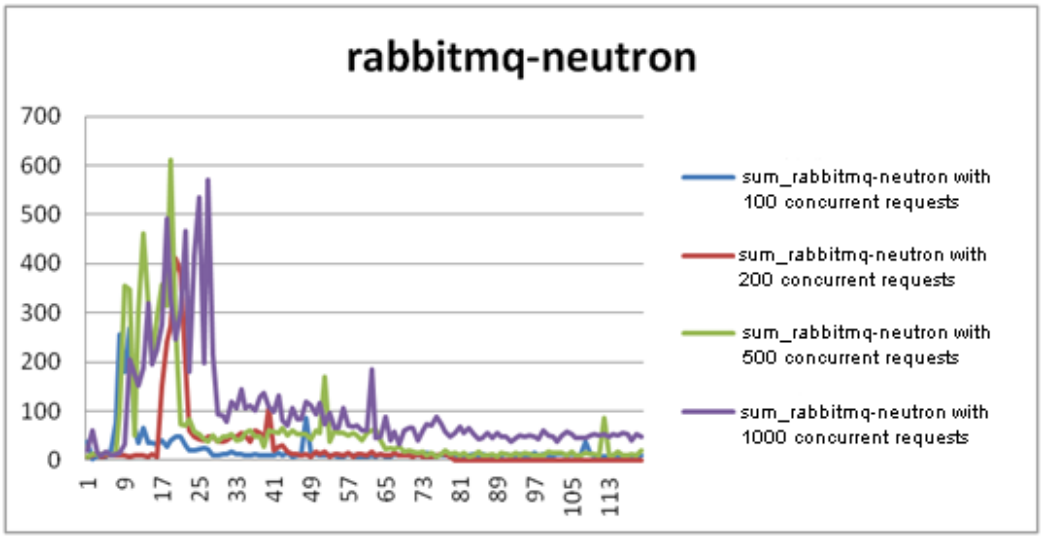
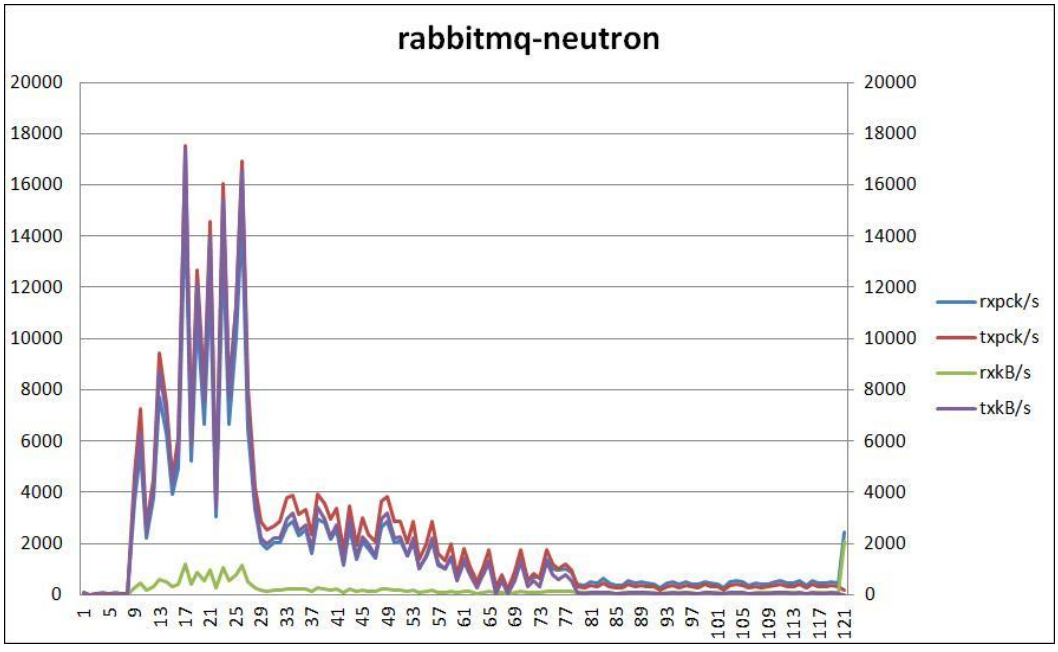


Figure 4-25 shows the network loads of the Neutron RabbitMQ node when instances are concurrently created.

Figure 4-25 Network loads of the Neutron RabbitMQ node during the concurrent instance creation



4.2 Performance analysis when system in heavy hybrid concurrency load test

4.2.1 Test Content

This test is conducted for 100 cascaded OpenStack systems accommodating one million instances to verify the hybrid concurrency load performance. The test focuses on the processing capabilities and resource consumption of all services at the cascading layer in hybrid concurrency scenarios. The following hybrid concurrency scenarios have been simulated to test the basic instance services:

- Concurrently creating 500 instances and deleting 500 instances
- Concurrently starting 500 instances and stopping 500 instances
- Concurrently creating, deleting, starting, stopping, and querying 200 instances

4.2.2 Test Result

According to the test result, all service requests in the hybrid concurrency scenarios are successfully processed, and the loads exerted towards the cascading layer are similar to those exerted in the single-service concurrency scenario. No performance risk is found.

Figure 4-26 shows the CPU usages of major services when instances are concurrently created and deleted.

Figure 4-26 CPU usages of major services when instances are concurrently created and deleted

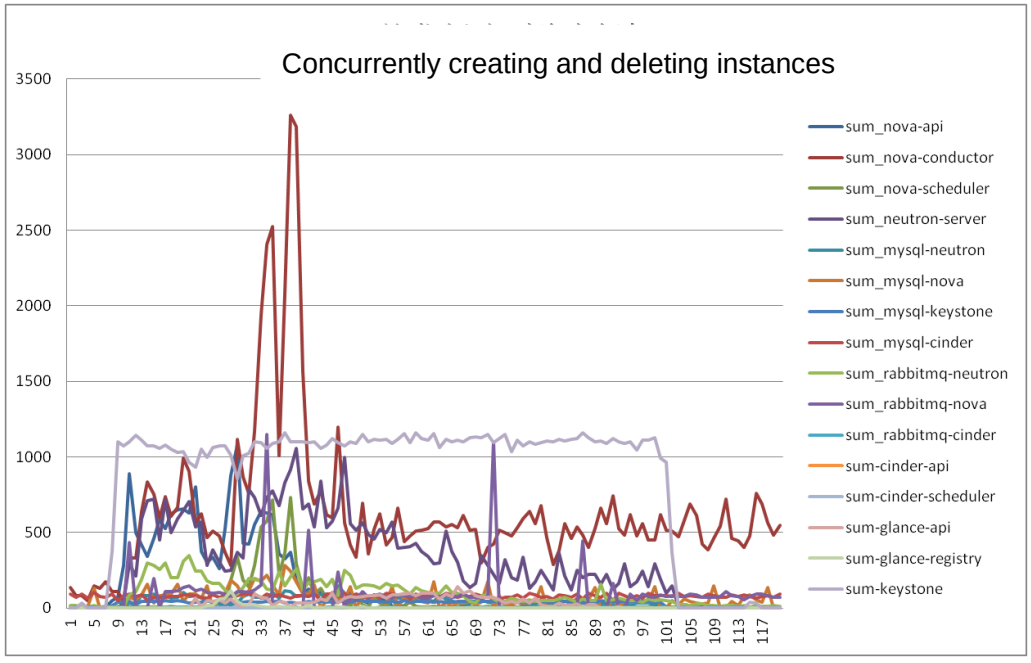


Figure 4-27 shows the CPU usages of major services when instances are concurrently started and stopped.

Figure 4-27 CPU usages of major services when instances are concurrently started and stopped

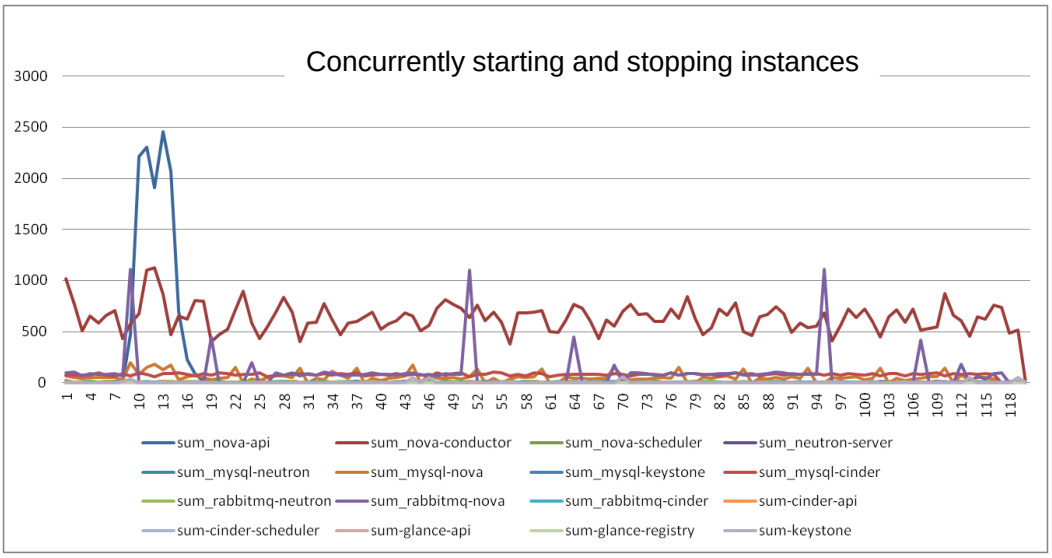
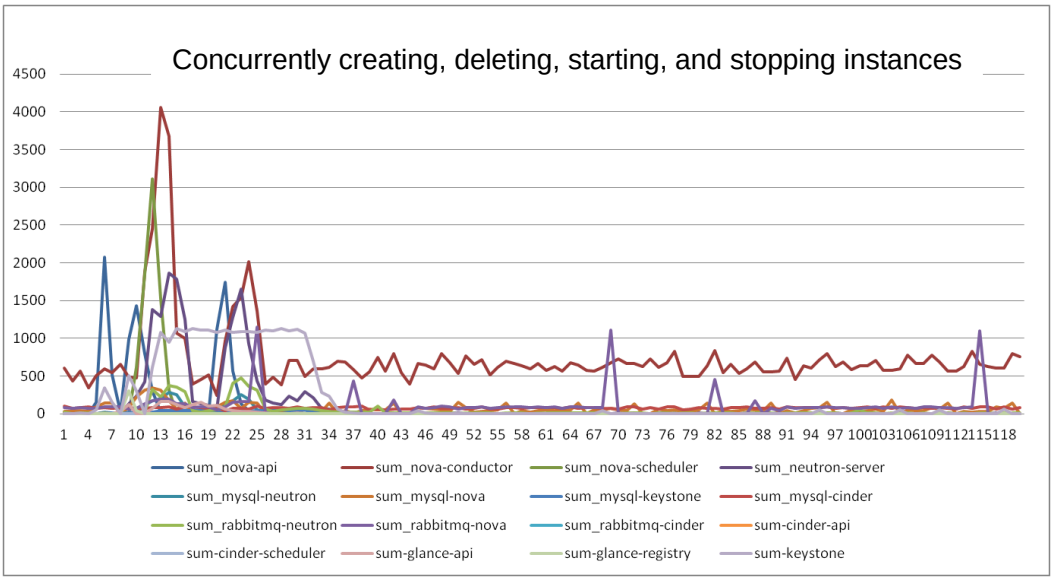


Figure 4-28 shows the CPU usages of major services when instances are concurrently created, deleted, started, and stopped.

Figure 4-28 CPU usages of major services when instances are concurrently created, deleted, started, and stopped



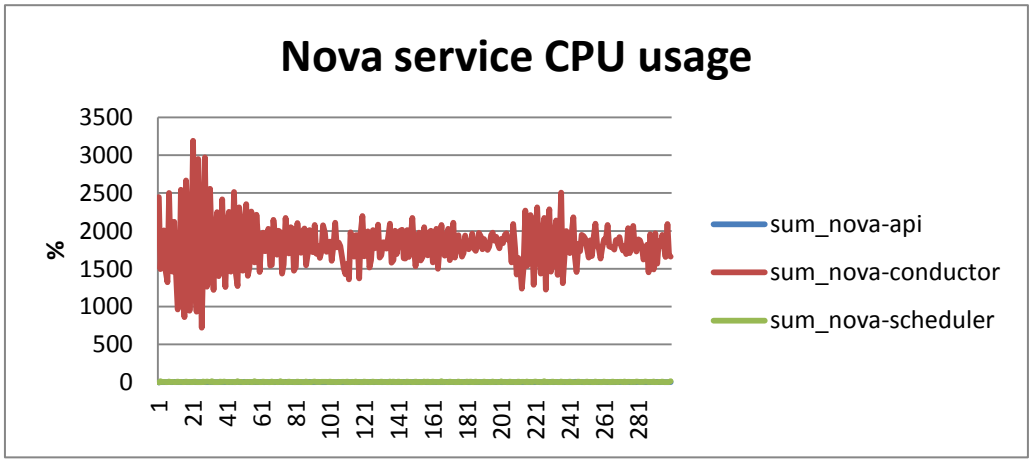
4.3 Performance analysis when system in idle-state

4.3.1 Nova

The Nova API processes (containing nova-api, nova-conductor, nova-novncproxy, and nova-consoleauth) are deployed on five servers, and nova-scheduler processes is deployed on three servers.

Figure 4-29 shows the static resource overheads for each Nova processes.

Figure 4-29 Nova CPU usages in a idle-state scenario with one million instances deployed

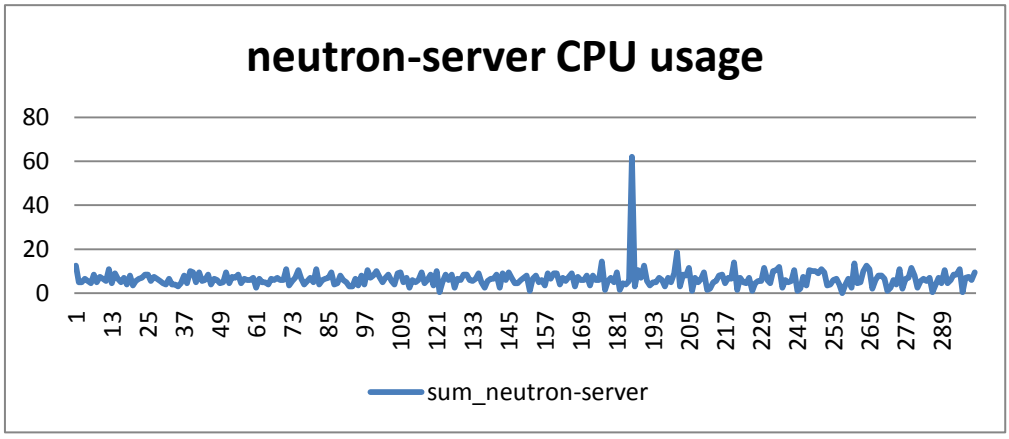


4.3.2 Neutron

The neutron-server process is deployed on six servers.

Figure 4-30 shows the CPU usages of the neutron-server process in idle-state scenario with one million instances deployed. The average CPU usage is lower than 10%, and the peak CPU usage is around 60%.

Figure 4-30 neutron-server CPU usages in idle-state scenario with one million instances deployed

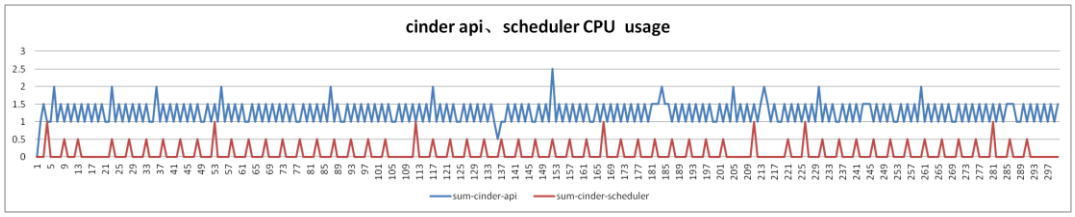


4.3.3 Cinder

cinder-api and cinder-scheduler are deployed on one physical server.

The CPU usages of both cinder-api and cinder-scheduler in idle-state scenario are lower than 5%.

Figure 4-31 cinder-api and cinder-scheduler CPU usages in a idle-state scenario with one million instances deployed

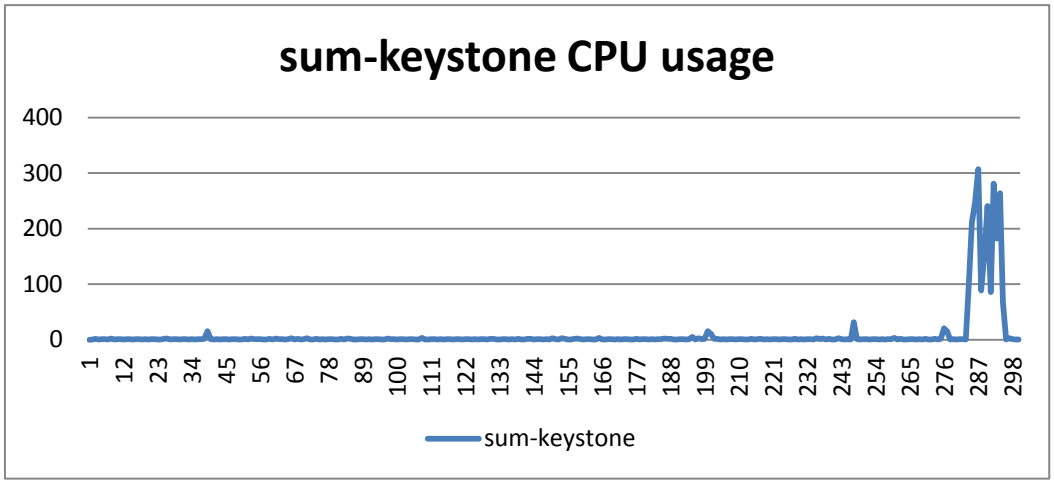


4.3.4 Keystone

The Keystone service is exclusively deployed on two servers

The average Keystone CPU usage in a idle-state scenario is lower than 30%, and the peak CPU usage in this scenario is around 300%. In the system of the current configurations, Converted the CPU usage to one server (RH2288 or E6000), the CPU usage will be lower than 30%..

Figure 4-32 Keystone CPU usages in a idle-state scenario with one million instances deployed



4.3.5 DB

Each of the Nova, Cinder, Neutron, and Keystone databases is exclusively deployed on one server.

In a idle-state scenario, the CPU usage of each database process is below 400%, and the peak CPU usage is around 500%. Converted the CPU usage to one server (RH2288 or E6000), the CPU usage will be lower than 30%.

Figure 4-33 Database CPU usages in a steady-state scenario with one million instances deployed

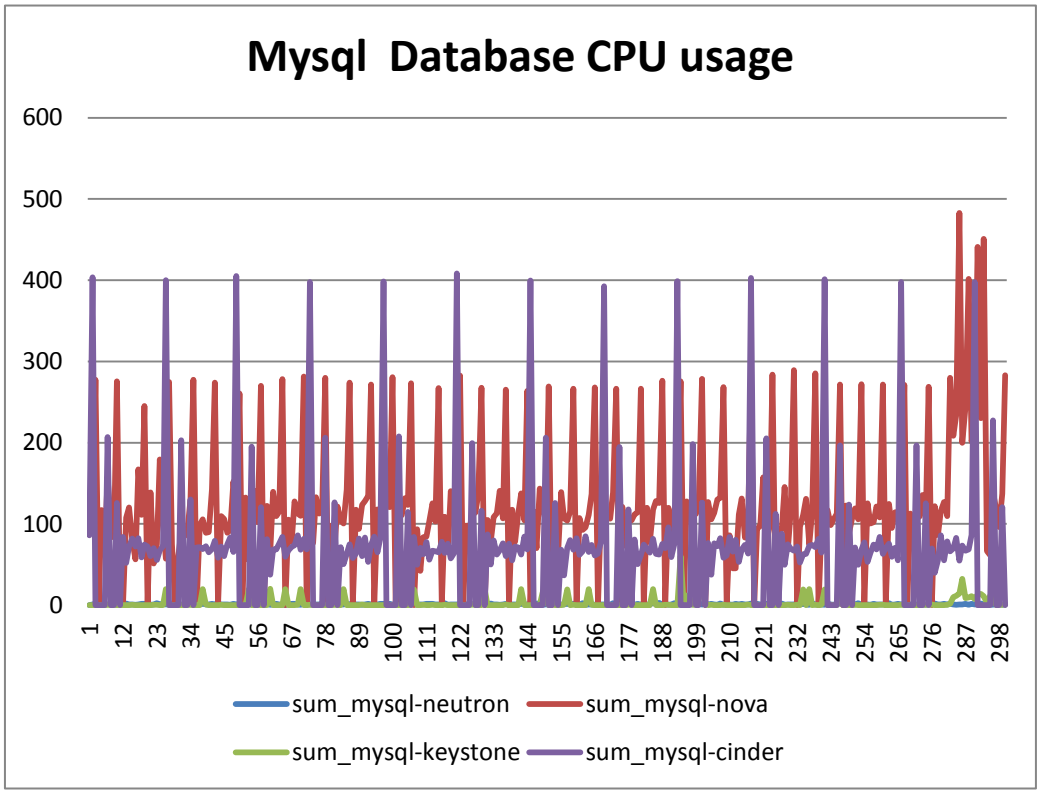


Figure 4-34 shows the Nova database I/O distribution in a idle-state scenario with one million instances deployed. The IOPS of database nodes is lower than 600. Therefore, RAID 10 array with 10 SAS disks can meet storage requirements.

Figure 4-34 Nova database IOPS distribution in a idle-state scenario with one million instances deployed

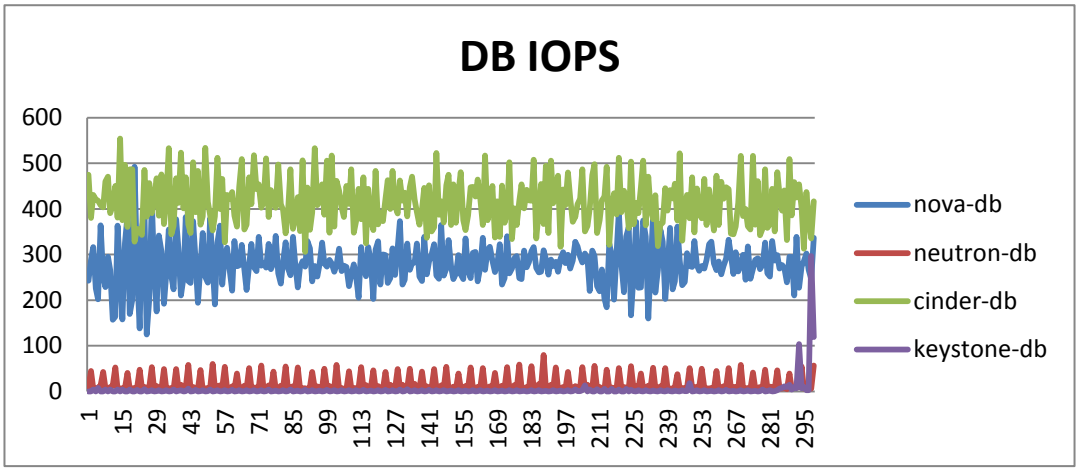
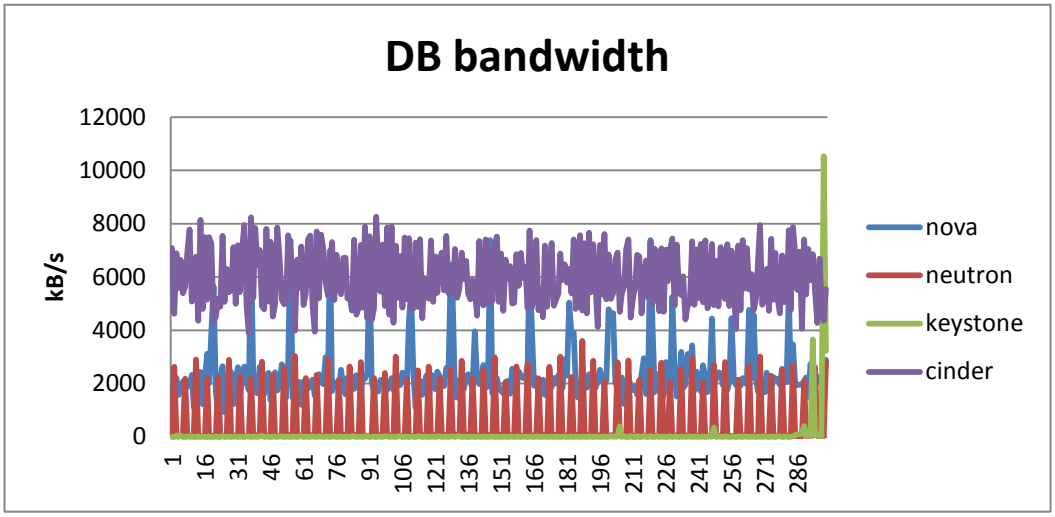


Figure 4-35 shows the Nova database bandwidth distribution in a idle-state scenario with one million instances deployed.

Figure 4-35 Nova database storage bandwidth distribution in a idle-state scenario with one million instances deployed

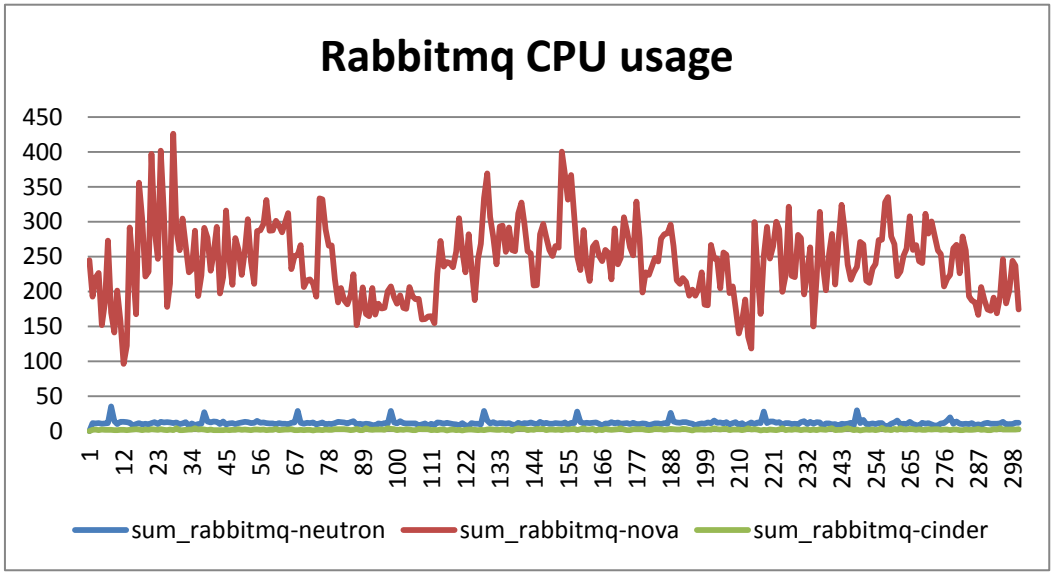


4.3.6 RabbitMQ

Each of the Nova, Cinder, and Neutron RabbitMQ processes is exclusively deployed on one server.

The CPU usage of the Cinder RabbitMQ and Neutron RabbitMQ is lower than 30%. Relatively, the Nova RabbitMQ is heavily loaded. The peak CPU usage of Nova RabbitMQ is about 400%. However, converted the CPU usage to one server (X6000 or E6000) , Neutron RabbitMQ CPU usage will be lower than 30%.

Figure 4-36 RabbitMQ CPU usages in a idle-state scenario with one million instances deployed



4.4 System Stability Test

4.4.1 Test Content

Create one million instances in batches when the cascading OpenStack system is running properly and have the created instances running stably for three days. Then observe the resource usages of services at the cascading layer and check for memory leaks.

4.4.2 Test Result

The following stability-related issues are found:

- On each node with the nova-api service deployed, a nova-api process consumes 17 GB memory. Memory leaks may occur.

158419	nova	20	0	209148	47828	1852	S	0.0	0.1	0:02.73	nova-api
158420	nova	20	0	414448	161952	4072	S	0.0	0.3	1:57.10	nova-api
158421	nova	20	0	374248	121660	4052	S	0.0	0.2	1:58.20	nova-api
158422	nova	20	0	17.702g	0.017t	4080	S	0.0	37.1	7:00.82	nova-api
158424	nova	20	0	374692	122128	4076	S	0.0	0.2	1:51.89	nova-api
158425	nova	20	0	374592	122016	4068	S	0.0	0.2	1:52.31	nova-api
158432	nova	20	0	228272	64664	1668	S	0.0	0.1	0:02.88	nova-api

- On the nodes with the Neutron server deployed, the neutron-server process consumes 26 GB memory. Memory leaks may occur.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
24194	neutron	20	0	31.771g	0.026t	1900	D	12.4	55.4	154:44.68	neutron-se+
24181	neutron	20	0	292796	9808	2572	S	0.0	0.0	0:11.23	neutron-se+
24188	neutron	20	0	293732	43268	2232	S	0.0	0.1	0:01.24	neutron-se+
24189	neutron	20	0	294472	44816	2272	S	0.0	0.1	0:01.43	neutron-se+
24190	neutron	20	0	293312	23316	2236	S	0.0	0.0	0:01.21	neutron-se+
24191	neutron	20	0	294852	44372	2272	S	0.0	0.1	0:01.64	neutron-se+
24192	neutron	20	0	302696	40240	2276	S	0.0	0.1	0:02.77	neutron-se+
24193	neutron	20	0	294924	28816	2256	S	0.0	0.1	0:00.81	neutron-se+
24195	neutron	20	0	294772	47444	2276	S	0.0	0.1	0:01.75	neutron-se+
24196	neutron	20	0	293668	44792	2272	S	0.0	0.1	0:01.35	neutron-se+
24197	neutron	20	0	294372	28188	2256	S	0.0	0.1	0:01.24	neutron-se+
24198	neutron	20	0	295320	48244	2276	S	0.0	0.1	0:01.46	neutron-se+
24199	neutron	20	0	300452	38236	2276	S	0.0	0.1	0:02.38	neutron-se+
24200	neutron	20	0	296080	51552	2140	S	0.0	0.1	0:54.42	neutron-se+
24201	neutron	20	0	296716	46412	2140	S	0.0	0.1	0:51.36	neutron-se+

4.5 Conclusion

According to the semi-simulation test in concurrent API request, and system load under idle state scenario in Phase I, the OpenStack cascading solution with current configuration can supports a maximum of one million virtual machines and is capable of handling 1000 concurrent API request.

Test of API request concurrency

Table 4-2 lists the CPU peak usage of major service processes in 1000 concurrent API request scenario.

Table 4-2 Peak CPU usage (%) of major service processes in a concurrency scenario (the cascading OpenStack)

Service	Service Process	Peak Usage(%)
Nova	Nova-api	2008.8
	Nova-conductor	5541.6
	Nova-scheduler	3309.3
Cinder	Cinder-api	1367.1
	Cinder-scheduler	93.9
Neutron	Neutron-server	2727.3
Keystone	Keystone	1213.4
RabbitMQ	Nova-rabbitmq	660.5
	Cinder-rabbitmq	68.4
	Neutron-rabbitmq	612.1
DB	Nova-db	445.5
	Cinder-db	208.6
	Neutron-db	344.3
	Keystone-db	70.4

The peak IOPS of the Nova database is 2400, that of the Cinder database is 1100, and that of the Neutron database is 920. Each database has a RAID 10 array consisting of 10 member disks, which are capable of handling such I/O requests.

System load under idle state scenario

Concluded based on the test data listed in Table 4-3, nova-conductor is under heavy compute loads, but all other services are running properly without overload. More nova-conductor should be deployed in production environment.

Table 4-3 CPU usage (%) of major service processes in a idle-state scenario (the cascading OpenStack)

Service	Service Process	Average Usage(%)	Peak Usage(%)
Nova	Nova-api	4.47	6.5
	Nova-conductor	1804.55	3189.1
	Nova-scheduler	5.85	17
Cinder	Cinder-api	1.25	2.5
	Cinder-scheduler	0.1	1
Neutron	Neutron-server	6.53	62.1
Keystone	Keystone	8.7	306.8

Service	Service Process	Average Usage(%)	Peak Usage(%)
RabbitMQ	Nova-rabbitmq	242.33	425.9
	Cinder-rabbitmq	2.36	4.5
	Neutron-rabbitmq	11.59	35.3
DB	Nova-db	122.87	482.7
	Cinder-db	70.34	408.5
	Neutron-db	1.18	5.9
	Keystone-db	2.41	60.4

In addition, the IOPS of each database in idle-state scenarios is lower than 600. Therefore, the system has no I/O risks.

System Stability Test

In the system stability test, memory leaks occurred on nova-api and neutron-server. Not find the root cause.

5 Phase II Test including L3 for the cascading OpenStack under the semi-simulated test bed

5.1 Test Content

This test is conducted for verifying the Neutron L3 network performance in different scenarios:

The following service models are available in the test:

- Small-scale tenants (4800): Each tenant has 4 networks and each network provide services for 10 instances. There are altogether 192,000 instances.
- Medium-scale tenants (150): Each tenant has 20 networks and each network provide services for 100 instances. There are altogether 300,000 instances.
- Large-scale tenants (50): Each tenant has 40 networks and each network provide services for 250 instances. There are altogether 500,000 instances.

The resources of about 100 small-scale tenants are scattered across four cascaded OpenStack instances, and other small-scale tenants each have only resources in one cascaded OpenStack. The resources of medium-scale tenants are scattered across five cascaded OpenStack instances, and those of large-scale tenants are scattered across ten OpenStack instances.

The test scenarios include:

- Create instances and routers and then concurrently add 500 router interfaces.

- Concurrently delete 500 router interfaces.

This test is conducted for 100 cascaded OpenStack systems accommodating one million instances.

5.2 Test Result

5.2.1 add_router_interface

Figure 5-1 shows the CPU usage of the neutron-server process.

Figure 5-1 neutron-server CPU usage

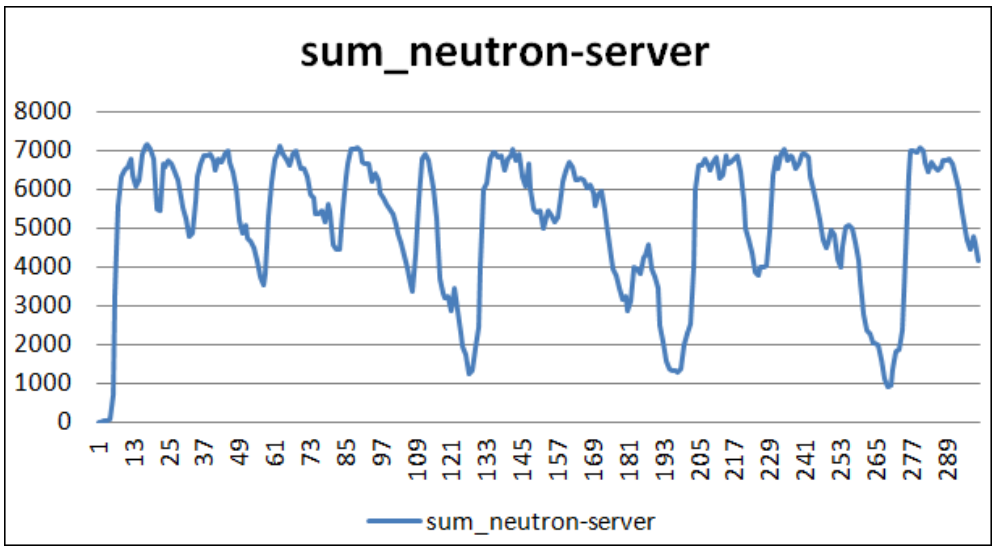


Figure 5-2 shows the CPU usage of the Neutron database.

Figure 5-2 Neutron database CPU usage

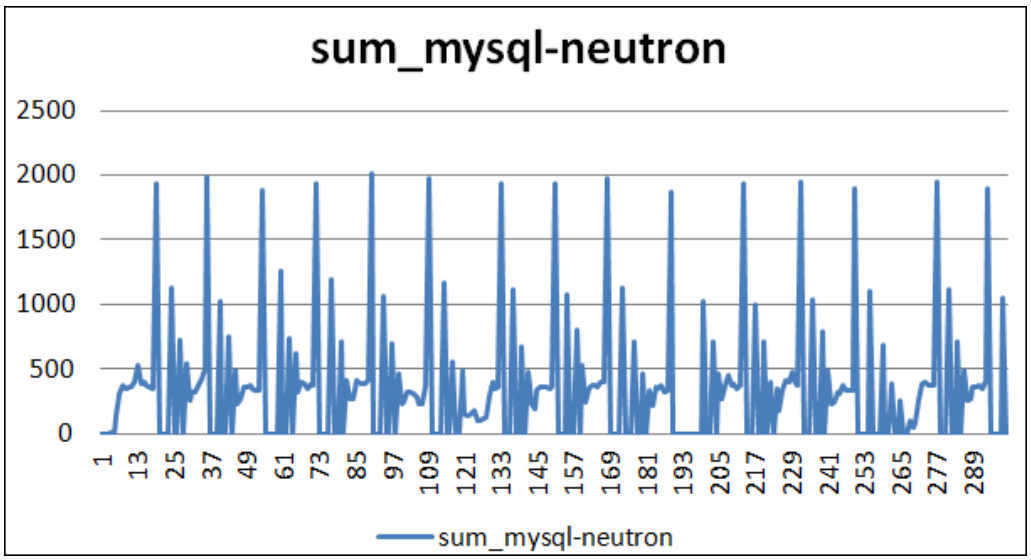
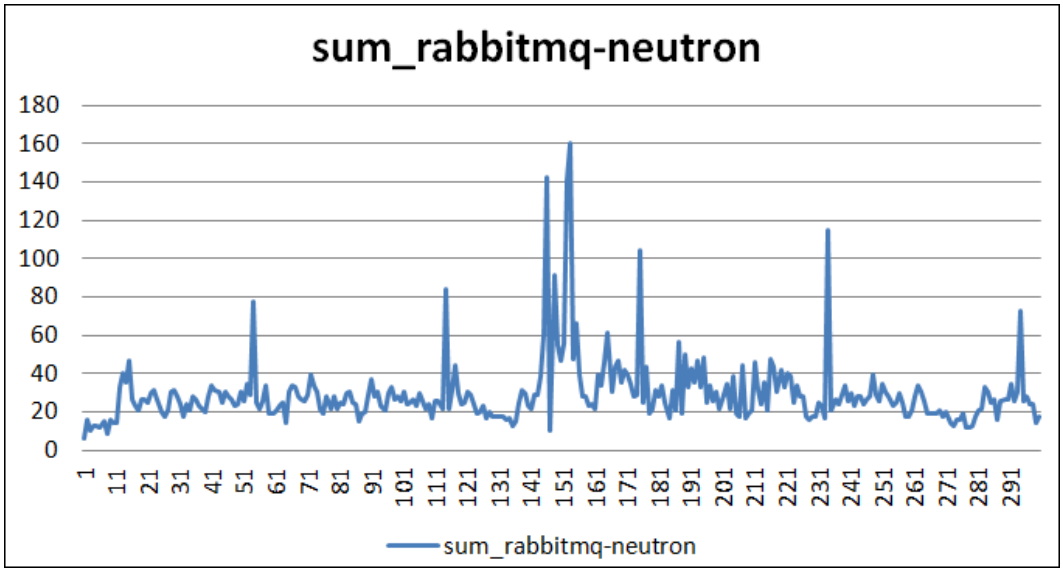


Figure 5-3 shows the CPU usage of Neutron RabbitMQ.

Figure 5-3 Neutron RabbitMQ CPU usage



5.2.2 remove_router_interface

Figure 5-4 shows the CPU usage of the neutron-server process.

Figure 5-4 neutron-server CPU usage

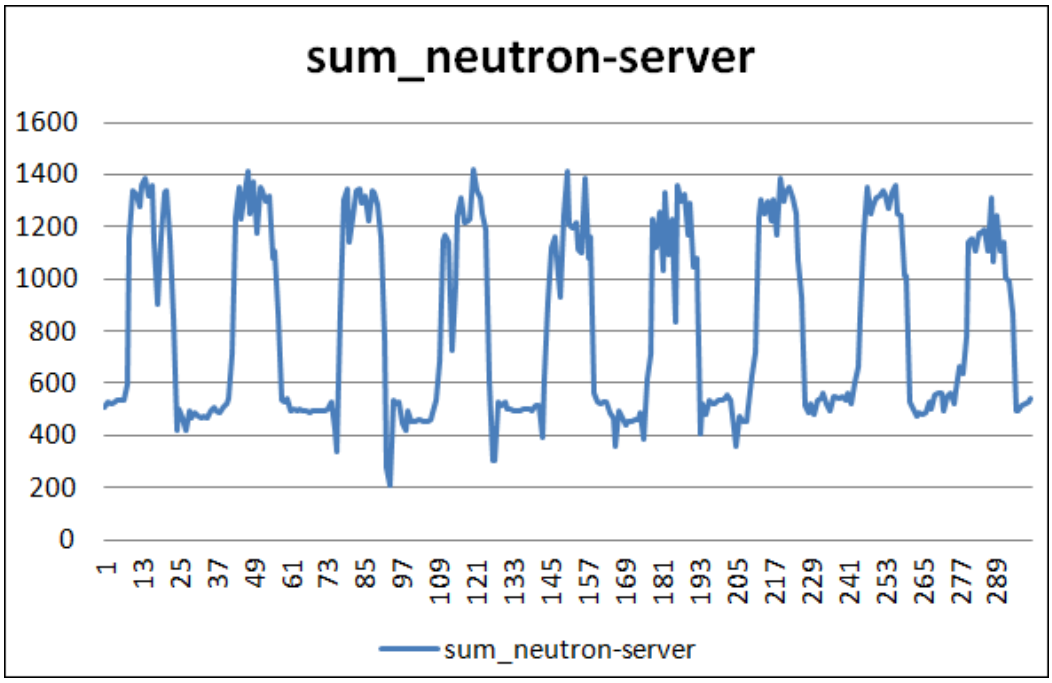


Figure 5-5 shows the CPU usage of the Neutron database.

Figure 5-5 Neutron database CPU usage

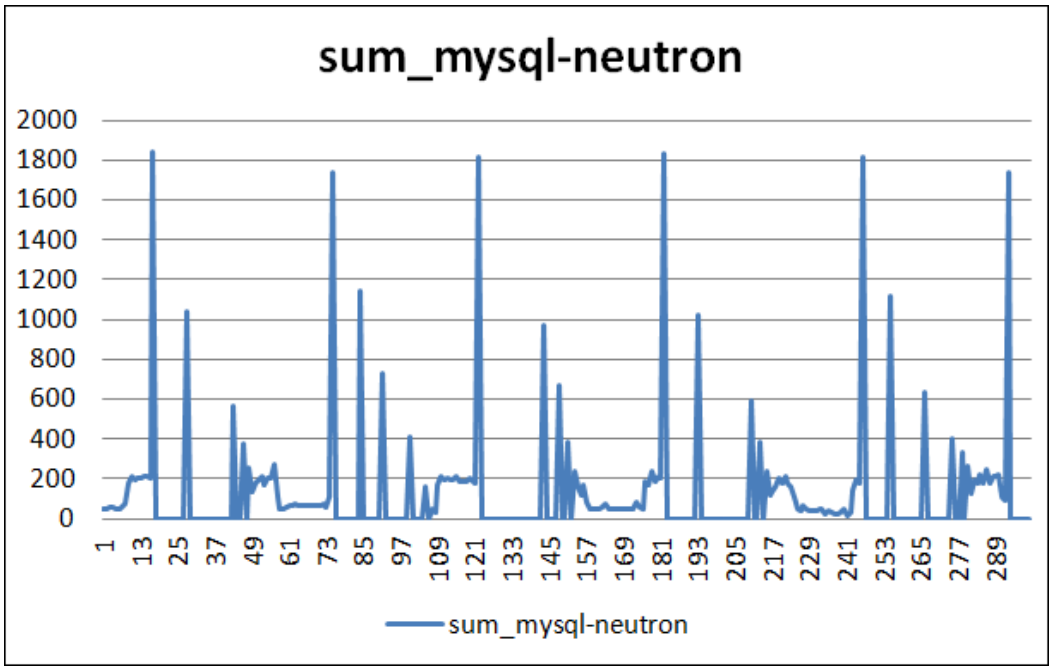
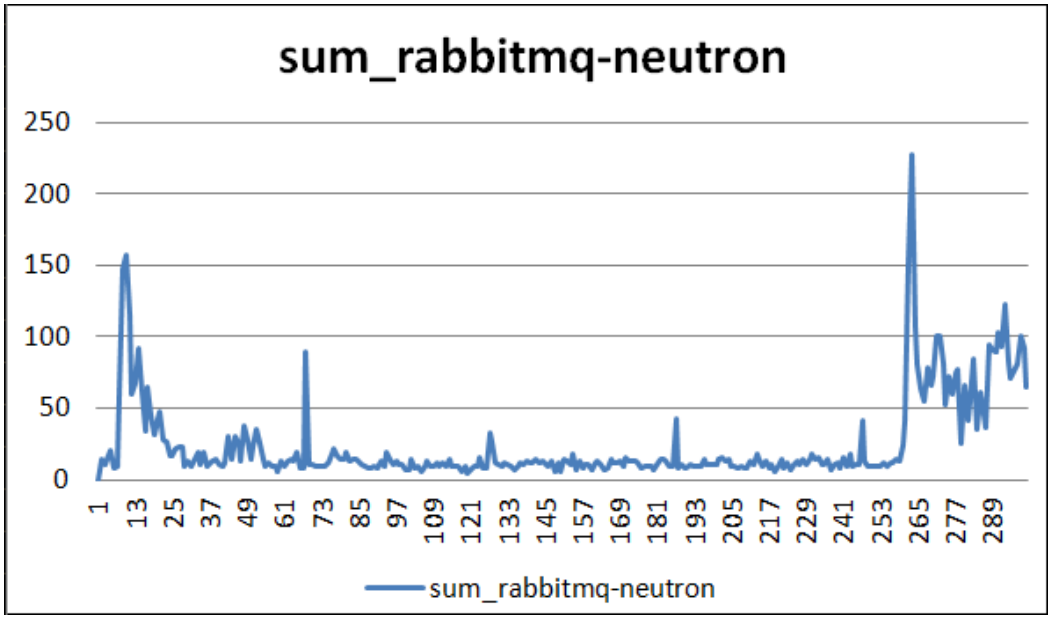


Figure 5-6 shows the CPU usage of the Neutron RabbitMQ database.

Figure 5-6 Neutron RabbitMQ database CPU usage



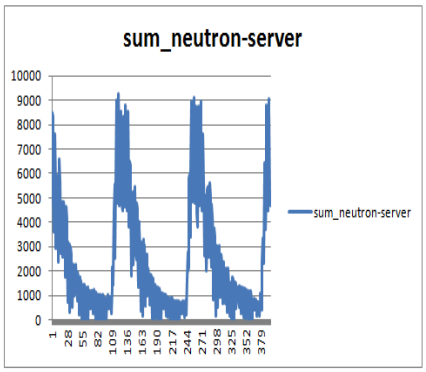
According to the test results, although the current L3 configuration meets the service concurrency requirements of all tenants, the neutron-server processes take on heavy workloads. Therefore, add more neutron-server nodes to share the workloads is recommended.

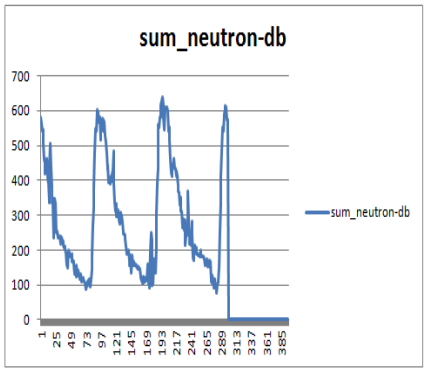
5.2.3 Instance creation concurrency after add_router_interface

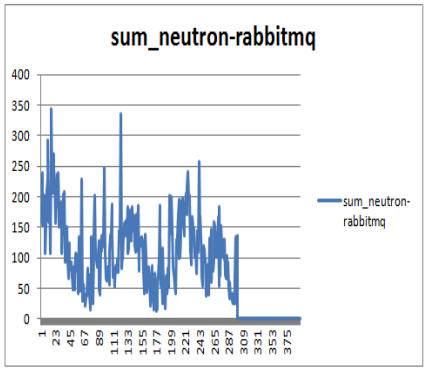
Sometimes, the network will be associated with router first, and then add new virtual machine to the network.

The test scenario is to concurrently create 500 virtual machines to networks which already have interface on the router.

The following picture shows the CPU usage of the Neutron RabbitMQ database.







5.3 Conclusion

During the phase II test, the CPU usage for Neutron-server, Neutron-DB and Neutron-rabbitmq is higher than that of Phase I. The test result shows that the OpenStack cascading solution with current configuration can supports a maximum of one million virtual machines and is capable of handling 500 concurrent API request.

Due to the lab has to be released for other assignment, the concurrency test for L3 has to be ended.

6 Issues found in the scalability test

During the test, some issues found and fixed, but better solution needs to be discussed in the community, some of them have been registered as bugs or BP.

6.1 KeyStone token size

6.1.1 Symptom

Adding 101 region to KeyStone, (1cascading OpenStack, 100 cascaded OpenStack. Authentication failure because of PKI token size is too large, uncompressed token, 536KB, compressed token, 40KB.

6.1.2 Why

More than 500 endpoints for 101 regions, the service catalog is very large in the PKI token, the original endpoint list occupied about 163KB.

6.1.3 Solution in the test

No catalog token. The better solution is to use Fernet token with distributed KeyStone servers. Discussion in the community: http://lists.openstack.org/pipermail/openstack-dev/2015-March/059148.html

6.2 Multi-worker Nova-Scheduler

6.2.1 Symptom

The resource usage of Nova scheduler is very high.

6.2.2 Why

Nova-scheduler doesn't support multi-worker mode, if Nova-scheduler is deployed on physical host, it's not able to leverage the power of mulit-core/multi-processor. Nova scheduler needs to support multi-worker mode. Otherwise has to add more physical servers to run Nova scheduler or run Nova-scheduler in virtual machines.

6.2.3 Solution in the test

Support multi-worker mode. BP registered: https://blueprints.launchpad.net/nova/+spec/scheduler-multiple-workers-support

6.3 Neutron port quota control

6.3.1 Symptom

During the batch instance creation process, the latency is significantly increased with the number of instances, or the instance creation fails due to the Neutron connection timeout. Especially for large project for example there are more 3k ports.

6.3.2 Why

During the instance creation, the nova-api queries network information from the neutron-server to validate networks and then query all ports of the tenant using the list_ports command to check whether the port quotas are sufficient. However, the list_ports command increases the workload of the neutron-server.

```
def validate_networks(self, context, requested_networks, num_instances)
    …
    if ports_needed_per_instance:
        ports = neutron.list_ports(tenant_id=context.project_id)['ports']
```

The quota control should be done in Neutron server, but not to query all ports and calculate the quotas in Nova.

6.3.3 Solution in the test

Commented the list_ports API request.

Better solution is to move the quota control to Neutron. To be discussed in the community.

6.4 Remove_router_interface makes Neutron server unresponsive

6.4.1 Symptom

The Neutron server becomes unresponsive when the remove_router_interface interface is invoked in large-scale deployment mode.

6.4.2 Why

This remove_router_interface first deletes the subnet ports of the router.After the deletion, the method check_ports_exist_on_l3agent and get_subnet_ids_on_router will call _core_plugin.get_ports, but the parameter in the calling is null, leads to query all ports,and twice during the remove_router_interface. As a result, the CPU usage of the neutron-server becomes too high for the system to respond.

6.4.3 Solution in the test

In the check_ports_exist_on_l3agent, if subnet_ids is null, just return false, to stop the get_ports query .

```
check_ports_exist_on_l3agent(self, context, l3_agent, router_id):
        subnet_ids = self.get_subnet_ids_on_router(context, router_id)
        if not subnet_ids:
            return False
```

Others also found this bug. https://bugs.launchpad.net/neutron/+bug/1420032

6.5 Neutron database CPU usage reaches over 2000%

6.5.1 Symptom

During the concurrent instance creation period, the CPU usage of the Neutron database reaches over 2000%.

6.5.2 Why

An analysis shows that three port query SQL statements take a long period of time to execute, more than 1 seconds. According to the ports table, the search criteria of these three statements are not indexed. Therefore, this issue can be resolved by adding three indexes to the ports table.

6.5.3 Solution in the test

Details are as follows:

Index 1: alter table ports add index tenant_device_id_idx (`tenant_id`,`device_id`);

Index 2: alter table ports add index network_id_mac_idx (`network_id`,`mac_address`);

Index 3: alter table ports add index network_id_device_owner_idx (`network_id`,`device_owner`);

Bug or BP will be registered soon.

6.6 VxLAN network concurrent creation failure

6.6.1 Symptom

Some VXLANs fail to create during the concurrent creation period..

6.6.2 Why

Some VXLANs fail to create because the allocated segment IDs are not locked. The success rate is about 20%.

6.6.3 Solution in the test

In /neutron/plugins/ml2/driver/helpers.py add lock in the function allocate_partially_specified_segment

```
with session.begin(subtransactions=True):
        select = (session.query(self.model).
                  filter_by(allocated=False,
**filters).with_lockmode('update'))
```

There is similar bug report on it: https://review.openstack.org/#/c/147540/

6.7 Timestamp based query for Neutron port status

6.7.1 Symptom

Low query efficiency to find out Neutron port status.

6.7.2 Why

Because the Neutron port status is changed by VIF plug and OVSAgent periodically detects, have to query Neutron port to find out the status of the port.

Currently Neutron does not support timestamp in Neutron port. No filter is available to query recently status changed ports.

6.7.3 Solution in the test

Add a timestamp and timestamp filter for Neutron port query so that we can retrieve port status changed during a specific period.

BP registered:
https://blueprints.launchpad.net/neutron/+spec/add-port-timestamp

6.8 Timestamp based query for Cinder volume status

6.8.1 Symptom

Low query efficiency to find out volume status.

6.8.2 Why

Because the Cinder volume creation is asynchronous operation, have to query Cinder volume to find out the status of the volume.

Cinder resources have created_at, updated_at and deleted_at attributes to record the operation time stamps. But currently we can only query resources operated at a given time stamp, or query all volume, then filter one by one to see volume status changed or not.

6.8.3 Solution in the test

Add a change-since filter for Cinder volume query so that we can retrieve resources status changed during a specific period.

BP registered:
https://blueprints.launchpad.net/cinder/+spec/resource-change-since-filter

6.9 Low query efficiency for Cinder snapshot and backup

6.9.1 Symptom

Low query efficiency for Cinder snapshot and backup.

6.9.2 Why

Currently, only cinder volume support pagination query by cinder Api,Cinder snapshot and backup doesn't support pagination query yet.

6.9.3 Solution in the test

Cinder snapshots/backup should also support pagination query, this is very useful and needed in large scale deployment of volumes application scenarios.

BP registered:
https://blueprints.launchpad.net/cinder/+spec/snapshots-and-bakcup-support-pagination-query

6.10 Suspicious NOVA-API 17GB memory usage

6.10.1 Symptom

If a large-capacity cascading OpenStack has been running for a long period of time, one of nova-api process on each node where the nova-api service deployed consumes 17 GB memory. Memory leaks may occur..

6.10.2 Why

Not found out why.

6.10.3 Solution in the test

This issue is resolved after the nova-api process is restarted

6.11 Suspicious Neutron-API 26GB memory usage

6.11.1 Symptom

If a large-capacity cascading OpenStack has been running for a long period of time, one of Neutron-server process on each node where the Neutron-server service deployed consumes 26 GB memory. Memory leaks may occur..

6.11.2 Why

Not found out why.

6.11.3 Solution in the test

This issue is resolved after the Neutron-server process is restarted

7 Acknowledgement

The test is executed, the report is produced by:

Cui Hongjuan, cuihongjuan@huawei.com

Yang Qingyue, yangqingyue@huawei.com

Chen Yan, sean.chenyan@huawei.com

Huang Chuibi, huangchuibi@huawei.com

Jia Hoajie，jiahaojie@huawei.com

Wei bo, bowei.wei@huawei.com

Jia Dong, jiadong.jia@huawei.com

Zhang Chi, z.zhangchi@huawei.com

Zhang Yu, zhangyu11@huawei.com

Huang Chaoyi, joehuang@huawei.com

8 Contact

The primary contact of this report is Huang Chaoyi. If you have any question, please send mail to joehuang@huawei.com