Tsinghua Science and Technology

Volume 21 | Issue 6 Article 3

2016

Cluster as a Service: A Resource Sharing Approach for Private Cloud

Donggang Cao
the Key Lab of High Confidence Software Technologies, Peking University, Beijing 100871, China.

Peidong Liu
the Key Lab of High Confidence Software Technologies, Peking University, Beijing 100871, China.

Wei Cui
the EMC corporation, Beijing 100027, China.

Yehong Zhong
the Key Lab of High Confidence Software Technologies, Peking University, Beijing 100871, China.

Bo An
the Key Lab of High Confidence Software Technologies, Peking University, Beijing 100871, China.

TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 03/09 pp610-619
Volume 21, Number 6, December 2016

Cluster as a Service: A Resource Sharing Approach for Private Cloud

Donggang Cao*, Peidong Liu, Wei Cui, Yehong Zhong, and Bo An

Abstract: With the rapid development of cloud computing and big data processing, an increasing number of application frameworks are being considered to run in a “cloud way”. This development brings about several challenges to the enterprise private cloud computing platform, e.g., being able to run most existing heterogeneous applications, providing scalability and elasticity support for newly emerged frameworks, and most importantly, sharing cluster resources effectively. In this paper, we propose a new service model, namely, Cluster as a Service (ClaaS), which is suitable for medium- and small-sized data centers to solve these problems in a relatively easy and general way. The idea behind this model is virtualizing the cluster environment for distributed application frameworks. Most applications can directly run in the virtual cluster environment without any modification, which is a great advantage. Based on lightweight containers, we implement a real system of ClaaS named Docklet to prove the feasibility of this service model. Meanwhile, we preliminarily design the definition of applications to make them easy to deploy. Finally, we present several examples and evaluate the entire system.

Key words: cloud computing; virtual cluster; Cluster as a Service (ClaaS); software definition

1 Introduction

At present, many enterprises are facing the problems of running an increasing number of application frameworks with limited resources. With the rapid development of cloud computing and big data processing, application frameworks for parallel computing are being developed continually. Although the newly emerged cloud computing and big data processing frameworks such as Hadoop, Spark, and Storm are becoming popular, many traditional distributed and parallel application frameworks such as MPI and OpenMP still play important roles. Almost every parallel application framework, whether traditional or not, requires a physical cluster to run effectively. However, most small- and medium-sized enterprises and organizations only have limited computing resources. They are unable to set up a cluster exclusively for every single application framework in need because these resources are costly and wasteful. Therefore, enterprises are in urgent need of an effective and easy approach of sharing a physical cluster among multiple frameworks to improve resource utilization. Simultaneously, different users in the same enterprise may demand separate replicas of the same framework. For example, a user needs to run some machine learning jobs on Spark but does not want to share an instance with others for privacy reasons; thus, multi-user and multi-instance support is also expected.

Several studies about sharing cluster resources among multi-frameworks have already been conducted. Mesos[1] is one of the most typical methods. However, Mesos is not flexible and not easy to support new frameworks. It requires programmers to write scheduling codes for the to-be-supported framework, which is a great burden and sometimes difficult task for non-experts. Moreover, in Mesos, one instance of each framework is typically shared by all users, which means

• Donggang Cao, Peidong Liu, Yehong Zhong, and Bo An are with the Key Lab of High Confidence Software Technologies, Peking University, Beijing 100871, China. E-mail: caodg@pku.edu.cn.

• Wei Cui is with the EMC corporation, Beijing 100027, China. E-mail: wei.cuiz2@emc.com.

∗ To whom correspondence should be addressed.

Manuscript received: 2016-06-30; revised: 2016-08-19; accepted: 2016-10-03

different users cannot run various instances. Moreover, framworks like Borg[2], Appollo[3], and Omega[4] are too specialized to manage private cloud of medium or small size.

Considering the problems above, this paper proposes Cluster as a Service (ClaaS), a resource sharing approach for private cloud environment. The main design goal of ClaaS is to build a scalable, efficient, and easy-to-use system to manage physical resources and provide virtual clusters as services to an enterprise's trusted users. Users can easily create a virtual cluster environment online and run applications such as Spark and MPI in this environment through WEB interface. ClaaS has the following capabilities and advantages. First, a reliable resource management and job management mechanism is needed. Second, the virtual cluster provides reliable communication methods between parallel processes inside to form a distributed entity. Third, the system automatically builds a virtual private network for distributed framework on demand. Finally, as the load of applications differs from each other, the virtual cluster of ClaaS is elastic to achieve high resource utilization.

According to the idea of ClaaS, we have developed a scalable lightweight cloud system called Docklet. Docklet takes advantage of Linux Container (LXC)[5], which is proven to be more efficient and appropriate to our requirements of ClaaS than other methods[6]. Software-defined technologies are utilized to help manage and virtualize cluster resources. ClaaS provides very simple interfaces for general purpose applications. Many applications have been deployed into the Docklet platform, such as Qemu VMs, VNC X11 Remote Desktop, Spark, Storm, MPI, Hadoop MapReduce, HDFS, Akka, Erlang, RabbitMQ, and Jupyter. All these applications can be accessed through WEB interface using a modern browser, which is convenient for enterprise users.

The rest of the paper is organized as follows. Section 2 provides some related work. Section 3 discusses the concept of ClaaS and beneficial points of constructing Vclusters. Section 4 describes the Docklet system based on ClaaS model. Section 5 explains implementation details of Docklet. Some examples and evaluations are given in Section 6 to demonstrate the rapid deployment of services and scalable effect of multi-host applications. Finally, we draw conclusions and explain future work in Section 7.

2 Related Work

Hindman et al.[1] presented Mesos, a task-oriented engine for resource allocation, to isolate many configurations at the IaaS layer. Mesos is beneficial for users to concentrate more on applications while limiting special infrastructural requirement. However, Mesos has difficulty in supporting new frameworks, and many additional steps are necessary to run something new on it. In the worst case, source codes of applications must be modified for platform adaption. Moreover, Mesos lacks multi-user isolation support, and it concentrates more on resource scheduling than offering cluster services. In our work, new frameworks can be easily supported through software-defined technologies, and we provide approaches for multi-user support.

Felter et al.[7] compared the performance of traditional Virtual Machine (VM) deployments and LXCs, and the researchers conclude that LXC equals or exceeds VM performance in all cases tested. In our work, virtualizing is based on LXC rather than VM for performance reasons. Although many public clouds use Docker in VM to obtain advantages of both methods, we attempt to solve the problems that might occur using LXC.

Doelitzscher et al.[8] described a purely cloud approach that provides on-demand high-performance computing for research projects and e-Learning in a private cloud. This approach can be extended to use Amazon's public cloud infrastructure as needed. Similarly, Brock and Goscinski[9] proposed the idea of exposing a cluster as a service in a cloud. Unlike in their work, ours focuses more on building an efficient and easy-to-use ClaaS platform for different frameworks. We also allow over-provisioning if some deployed virtual clusters are not in use.

Openstack[10] is a cloud operating system to allocate OS environment of node-unit, which is compatible with numerous mainstream applications, without the need to modify their source codes. However, this approach lacks efficiency and scalability for many distributed and elastic frameworks. It works at the IaaS level, so it is not friendly to distributed application frameworks. Moreover, building, managing, and using a private cloud platform via tools like Openstack is a heavy and difficult task. These tools come without any cluster management methods as we supposed.

3 ClaaS and Vcluster

ClaaS is different from the single-node service provided by IaaS and the application interfaces to programs provided by PaaS. It describes a general method to define applications of different types, achieving good compatibility, efficiency, and scalability for applications on enterprise private cloud.

3.1 Vcluster and why it is suitable

Virtual Cluster (Vcluster) provides an integral view of a logical cluster for its owner to control exactly one kind of application framework. Every Vcluster consists of a series of distributed and isomorphic nodes. Thus, ClaaS provider can dispatch different applications separately into different Vclusters. The design of such Vcluster has the following beneficial characteristics:

Flexibility: Each node in Vcluster provides an environment similar to traditional VM, and new frameworks can be easily supported and extended simply by creating new Vclusters.

Extensibility: Multiple nodes in the same Vcluster can be naturally executed on different machines to make use of multiple physical resources for a single framework.

Scalability: Each Vcluster allows the applications inside to control the life cycle of working nodes conveniently, which is beneficial for frameworks to schedule for the elastic use of resources.

Definability: A Vcluster can be described and reconstructed using software definition technology. Owners can define features including distributed architecture, migration policies, and execution scripts, thereby demonstrating good flexibility. Moreover, this procedure is independent of hardware topology, as depicted in Fig. 1.

As Fig. 1 shows, this kind of service model provided by ClaaS is beneficial for users to operate in an IaaS environment. It can also be used for installing distributed frameworks as PaaS mode.

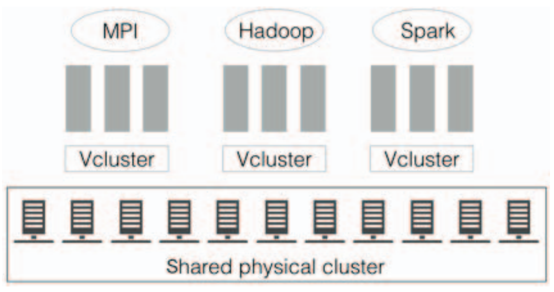


Fig. 1 Multiple Vcluster for multiple tenants.

3.2 Software defined technology toward ClaaS

Several technologies that are emerging in virtualization promote the generation of software definition, which brings about an improvement in constructing devices by feasible software protocol instead of fixed hardware topology. These technologies include Software Defined Network, Software Defined Storage, and Software Defined Datacenter.

Software definition is applied to ClaaS, and it is named Software Defined Vcluster. Software Defined Vcluster requires users to define Vcluster with several options that are sufficient for ClaaS provider to build logical clusters, handle multiple cases of cluster status, and execute internal applications.

The advantage of Software Defined Vcluster consists of the following two aspects:

Reducing the complexity of cluster deployment. Users need not set up nodes manually but simply provide simple policies about what each Vcluster should do to handle certain cases such as scaling, recovery, and migration.

Simplifying the management of distributed services. Vcluster and applications inside can be shared by multiple users and coherently rebuilt at any time, which reduces the burden of both administrators and users.

4 Design of Docklet

Docklet is a system designed to support ClaaS. It is now open source on Github. We begin the description of Docklet by explaining its target environment and discussing the design goals of Docklet and Docklet architecture.

4.1 Target environment and problems

Docklet is designed for enterprises with a medium- or small-scale data center, and it is used mainly by trusted enterprise employees.

Take Software Engineering Institute (SEI) of Peking University as an example. The data center in SEI consists of about 10+ high-performance computers and a high-speed LAN. Staff and students make use of them for various purposes, such as web portal servers, machine learning tasks, distributed algorithm experiments, and other diverse programs running on them. Some are long-term jobs that require a stable environment, whereas some are for experiments that run occasionally.

Currently, the SEI data center uses IaaS for resource sharing, which has some problems. To meet the requirements of all the abovementioned diverse jobs, VMs are used to share the physical resources. Users have their own VM on physical servers. Therefore, data center administrators often have to deal with problems like reinstalling the system and reallocating VMs. Users also complain of many issues such as having to manually set up the environments for distributed applications and resource shortage.

Therefore, we design Docklet to provide better resource sharing than the current IaaS approach.

4.2 Design goals

Docklet aims to provide robust and elastic Vcluster services on small-sized data centers for researchers or employees. Given that these users are used to working in the environment of physical machine or VM and will not spend an extensive amount of time for learning to use a new ClaaS system, we aim to provide logical/virtual clusters that can be used in the traditional way that users are familiar with. Some convenient features are also introduced in Docklet. For instance, with definitions in software, users can build a specific Vcluster within a very short time and execute services or frameworks automatically as soon as the Vcluster starts up. Low consumption of physical resources and high performance are demanded, so we turn to LXC to pursue characteristics that traditional VMs like KVM and XEN have difficulty in achieving. LXC is also simple to use and manage compared with other container technologies such as Docker.

Elasticity of Vclusters in Docklet is required for distributed programs or frameworks such as Hadoop and Spark. Users or programs inside the Vcluster can control the scale of Vcluster in our design. To open Docklet for extension and supervision, Docklet should provide an easy-to-use interface. We use HTTP protocol in the current version so that only web browsers are needed to access Docklet services.

4.3 Overview

A user view of Docklet is shown in Fig. 2. Users can access resources via a workspace, which is the user's working space and mainly consists of applications and data. Users perform jobs such as running applications and analyzing data in the workspace. Each workspace is supported by one Vcluster. Vcluster is a unit for

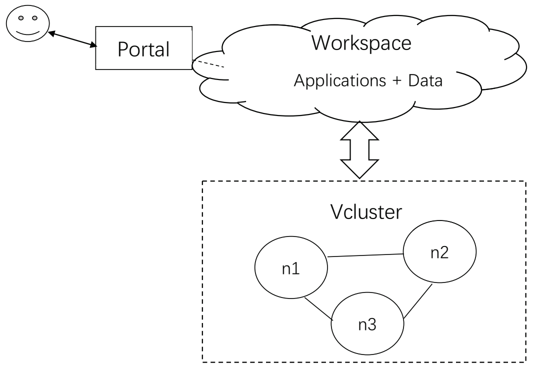


Fig. 2 User's perspective of one Vcluster and workspace.

resource management. A Vcluster consists of some virtual nodes and a private network. Users can manage their jobs by creating, restarting, and scaling in and out the Vcluster in browsers. Moreover, each workspace provides a portal for users to perform operations on nodes. Nodes are isolated from each other, and only nodes from the same workspace are connected by the network, that is, nodes in one workspace cannot affect nodes in other workspaces. A virtual node is LXC.

Nodes in Docklet are created from node images, which are similar to VM images or Docker images. Docklet has a simple but powerful mechanism to build, share, and use images for different enterprises. When we want to support a new framework in Docklet, we can create the appropriate environments of that framework in a virtual node and then save the node snapshot as images. If we select to share this image, all users in Docklet can see it and make use of it, which is highly convenient.

Figure 3 shows the main components of Docklet. We use a web server to show users a graphic interface and an http server to handle requests. Docklet Core is the backend. Docklet Core consists of a master process that manages worker daemons running on each cluster node and records the status of all Vclusters, physical nodes, and users.

The master handles requests and decides whether the Vcluster should be deployed and where it should be located in consideration of the status of physical machines and workload of existing Vclusters. Once the master decides to build a Vcluster, it passes the message to a proper worker to start the container of the specified image. Subsequently, network topology of this Vcluster will be constructed, and communications between containers in a Vcluster will be guaranteed

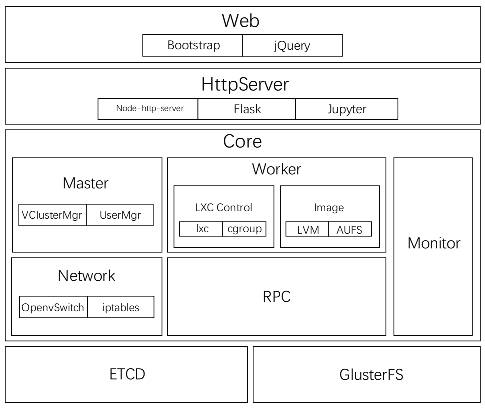


Fig. 3 Architecture of Docklet.

by the network component. We use LVM and AUFS to store snapshots (or images) of containers so that containers can be easily saved, replicated, and shared to other users. Simultaneously, the status of physical resources and Vclusters can be seen by the monitor.

With Docklet, users can create a dynamic elastic cluster through web browsers with software environment ready to run their tasks (for example, start to run Spark tasks without any configuration). The cluster can be stopped or restarted at any time and shared to others. When tasks are running, users can see the status of the cluster in a webpage. Once the tasks are finished, the cluster can be destroyed and resources are released for new clusters.

Docklet is easy to use for administrators. Docklet can be installed and executed in only several instructions because many tools are used for simplifying this complex task. Moreover, regardless which node the user is controlling, the user interfaces (which will be discussed in the next section) can be identically invoked to obtain or set the properties of Vcluster, such as querying for the number of nodes in Vcluster.

Docklet provides a portal externally for each workspace to allow users from an external network to interact with a certain node directly. We use Jupyter, an open-source project, to achieve functions of controlling workspaces in browsers. Docklet is designed to make every node in one Vcluster completely connected in a sub-network, so users have access to every node by visiting the portal.

In summary, Docklet system has the following characteristics.

Isolation: Docklet uses multiple measures to keep nodes away from incidents that can harm other nodes and the external environment, including CGroup, Capacity Drop, Namespace, NetFilter, Apparmor, and Filesystem Quota.

Efficiency: One physical machine can be shared by nodes in Vclusters for multiple users, which is beneficial for the efficient use of resources provided by a limited number of machines.

Simplification: Docklet is highly convenient for users to deploy a cluster-based application quickly because it performs many complex and tedious configuration jobs for each Vcluster automatically, such as sub-network connection, data sharing spaces, and allocation of open-ssh keys for authorization.

Transparency: Regardless which machine a virtual node is laid on, the perspective from the internal environment of this node is no different in terms of network IP, root file system, and shared data, even after the node is migrated from one machine to another.

5 Design of Docklet

5.1 Interface

For ease of introspection and open extension, Docklet API is designed in a RESTful style. Flask is used to handle HTTP requests and reduce work such as token management. When a user tries to login Docklet, a token will be generated for the following instructions and it will expire after some time. All HTTP requests sent to Docklet master shall contain a valid token or they will be ignored. Docklet master deals with all requests and may send instructions to Docklet workers when necessary.

Authentication of users is conducted by a module named UserManager, which uses Flask-SQLAlchemy, a package of Flask, to store user information. Additionally, PAM authentication is available, and we provide a plugin to enable external login methods. For example, IAAA authentication of Peking University can be used in our instance of Docklet.

All APIs are carefully designed to describe their meaning precisely. Some important APIs are listed in the table below. In these APIs, response content types are all “application/json”.

```
POST   /cluster/create/
Create an original workspace/Vcluster for
a specific user
Parameters
{
    "token": "string",
        //indicating the user’s information
    "clustername": "string",
        //name of the workspace to be created
    "imagename": "string",
        // name of the image that will be used
    "imagetype": "string",
        //type of the image
    "imageowner": "string",
        //owner of the image
}
Response Class (Status 200)
{
    "success": "string",
        //indicating if the instruction succeeded
    "message": "string",
        //error message
}
```

```
POST   /cluster/start/
Start all containers in a workspace
Parameters
{
    "token": "string",
        //indicating the user’s information
    "clustername": "string",
        //name of the workspace to be started
}
Response Class (Status 200)
{
    "success": "string",
        //indicating if the instruction succeeded
    "message": "string",
        //error message
}
```

```
POST   /cluster/save/
Duplicate a container?s content to an image
Parameters
{
    "token": "string",
        //indicating the user’s information
    "clustername": "string",
        //name of the workspace that
        // the container is in
    "image": "string",
        // name of the image that will be created
    "description": "string",
        //description of the image
    "containername": "string",
        //name of the container
    "isforce": "string",
        //if this instruction will be executed
        // by force(with a -f flag)
}
Response Class (Status 200)
{
    "success": "string",
        /indicating if the instruction succeeded
    "message": "string",
        //error message
}
```

```
POST  /monitor/vnodes/<con_id>/<issue>/
Fetch the current status of a node
Parameters
{
    "token": "string",
        //indicating the user’s information
        //"con_id" is the identifier of container
        //"issue" is "cpu_use" or "mem_use" or
        //  "disk_use" or "basic_info" or "owner"
}
Response Class (Status 200)
{
    "success": "string",
        //indicating if the instruction succeeded
    "moniter": "string"
        // the result
}
```

Docklet uses ETCD to monitor the healthy status of physical machines (see Host-X and Y in node pool from Fig. 4), and each machine holds a partial set of nodes from Vclusters of multiple users. Critical data in the system are stored in the Key/Value database provided by ETCD.

User’s data are stored in an NFS cluster that is independent of Docklet system. The sharing of data between NFS cluster and Docklet nodes is through a high-speed network. Each user will have his or her own storage share in the NFS cluster. The user’s storage will be automatically mounted into Vclusters so that the user can manage data conveniently.

5.2 LXC Controller and Image Manager

Docklet introduces a software definition approach to build Vclusters. In particular, users can write certain configurations and submit them to Docklet to build a Vcluster with specific content and behavior. Moreover, containers can be easily migrated among servers in Docklet. This approach is implemented in LXC Controller and Image Manager.

LXC Controller is a module that manages LXC behaviors such as starting, stopping, and rebooting containers. This module is inside Docklet worker and will be invoked when Docklet master sends instructions of manipulating containers. Containers in Docklet have three statuses (Fig. 5), namely, running, breakdown, and

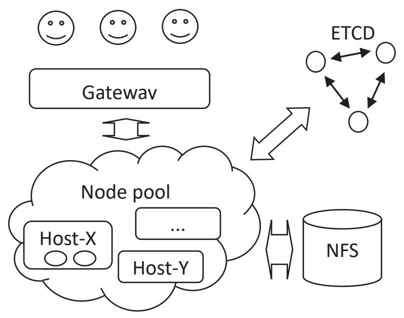


Fig. 4 Architecture of Docklet implementation.

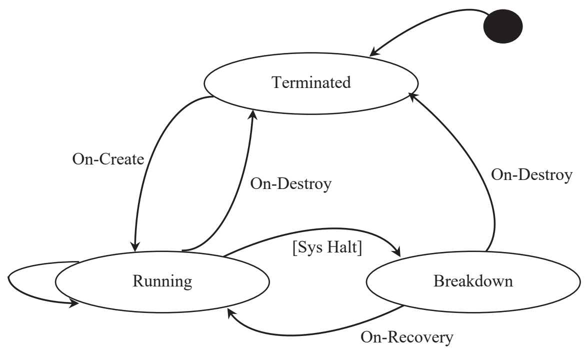


Fig. 5 State diagram of a node.

terminated. When a container in a Vcluster is created, LXC Controller will check if this user has sufficient resource share to support this container. If this machine has enough resource, the LXC configuration is set for LXC core. On-Create scripts are triggered inside a container once launched, and On-Destroy scripts are triggered before a container is to be destroyed. On-Recovery scripts are responsible for handling abnormal breakdown and turning it into normal running status, which is useful for certain applications to conduct journal recovery or dirty file cleaning.

In Docklet, the disk storage of a container and its configurations can be saved as an image. Image Manager is designed to store and share these images among users. Root-FS of every container is based on Union-FS[11], similar to Docker10, and consists of multiple Read-Only Layers and single Read-Write Layer. Docklet sets a volume quota to Read-Write Layer for security consideration.

Given that containers can be easily replicated from images in Docklet, we develop an approach for sharing images so that newcomers can easily take advantage of others' work and skilled users can promote their work. For example, a user can start a Spark Vcluster using images made by another user and start to run machine learning programs on it even without ideas of how to deploy Spark on a normal cluster. The only operation needed is to select the image and start a Vcluster with it. This technique is convenient for many people unfamiliar with a specific framework.

The Image Manager will stop a container, replicate its content and configuration when a saving instruction is sent, and transfer it to a special storage linked to a private repository for storage. When a user selects to share an image in his/her private repository, a copy of this image will be shown in the public repository so that everyone can use it.

5.3 Network

The network module of Docklet is mainly based on OpenVSwitch, an open-source implementation of OpenFlow[12] technology, and some iptables configurations. This module is used to provide a transparent and virtualized sub-network among nodes in the same Vcluster, so that containers of a Vcluster are in a real LAN. A network config template in LXC is shown below.

```
Network config template in Linux container
##### config #####
lxc.include = /usr/share/lxc/config/ubuntu.common.conf
lxc.rootfs = /var/lib/lxc/base/rootfs
lxc.utsname = base
lxc.network.type = veth
lxc.network.name = eth0
lxc.network.link = docklet-br
#lxc.network.veth.pair = baselink
lxc.network.ipv4 = 172.16.0.10/24
lxc.network.ipv4.gateway = 172.16.0.1
lxc.network.flags = up
lxc.network.mtu = 1420
```

A network structure of Docklet is shown in Fig. 6. We build GRE tunnels between hosts to connect containers together without extra network devices. When a Vcluster is created in Docklet, a sub-network will be built, and containers will be generated under this sub-network. Public and private SSH keys will also be generated and copied to each container to achieve logging without a password.

6 Experiment and Evaluation

To show the capabilities and effects of Docklet, we perform several experiments on our physical machines. All of the examples and evaluations are based on our Docklet system, which is set up into a cluster composed of three physical machines. Each machine has 400 GB physical memory and 32-core (Xeon E5-2670 2.60 GHz) CPU. Docklet makes use of CGroup

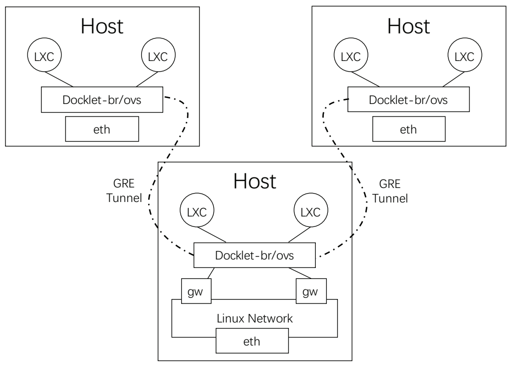


Fig. 6 Docklet network on multi-hosts.

to limit every LXC to use at most 1 CPU and 4 GB memory. Extra three machines, each of which contains 3 TB RAID5 Harddisk, are established as global NFS.

6.1 Example: Spark jobs

This example describes how to deploy Spark on Docklet and run tasks on it. The Vcluster has three nodes. The names of the three nodes are host-0, host-1, and host-2 by default. Spark jobs can be initiated by only a few scripts.

```
Start scripts for Spark
--------------------------------------------
root@host-0:~# cd /home/spark
root@host-0:/home/spark# ./sbin/start-master.sh
root@host-0:/home/spark# ssh root@host-1 \
/home/spark/sbin/start-slave.sh spark://host-0:7077
root@host-0:/home/spark# ssh root@host-2 \
/home/spark/sbin/start-slave.sh spark://host-0:7077
root@host-0:/home/spark# ./bin/spark-submit \
--master spark://host-0:7077 examples\
/src/main/python/pi.py 10
```

6.2 Example: MPI jobs

MPI jobs could also be launched easily. The following commands show an example of compiling and running an MPI program in the three nodes of the Vcluster.

```
Start scripts for mpi
--------------------------------------------
root@host-0:~# cd /nfs
root@host-0:/nfs# mpic++ ./program.cpp -o mpi.bin
root@host-0:/nfs# mpiexec -hosts host-0,host-1,\
host-2 -n 6 ./mpi.bin >/nfs/result.out
```

6.3 Example: Scalability of I/O jobs

We evaluate the performance of a Map-Reduce Job in Docklet system in this experiment. The experiment performs a parallel word counting algorithm on files of 14 GB data stored in GlusterFS, and data transmitted between GlusterFS and Docklet nodes are completely through TCP/IP network. Table 1 shows an improved speed-up of I/O jobs.

6.4 Example: Elasticity of Spark GraphX jobs

We conduct a Spark PageRank job in Docklet to verify its elasticity performance. Considering that the calculation complexity of every super-step in PageRank is equivalent and stable, the performance of each super-step varies from time to time according to the change in number of working nodes.

The elasticity manager (invoking scaling in or out at a certain second) in this job will change the number of nodes with time (Fig. 7), and this chart shows how the number of nodes in Vcluster influences the performance of PageRank job. In spark service, a newly connected spark slave usually takes around 10 s to initialize itself and then prepares to boost the performance of jobs. Meanwhile, releasing a spark slave would immediately slow down its performance.

Table 1 Speed-up of I/O jobs with scalable working nodes.

NFS text (GB)	Serial (min)	24 nodes (min)	Speed-up
14	14	1.6	8.75
37	32	3.0	10.67

6.5 Network efficiency

All our facilities and statistical results are based on traditional routers and switches, and no advanced hardware is deployed inside our Docklet system. With the topology shown in Fig. 8, we verify the network throughput between every pair of nodes, each of which is either a physical host or a container. In addition, network bridge of each container is established by OpenVSwitch (v2.4.0) GRE Tunnel, called ovs-bridge, and the MTU value of OpenVSwitch devices is set to 1420.

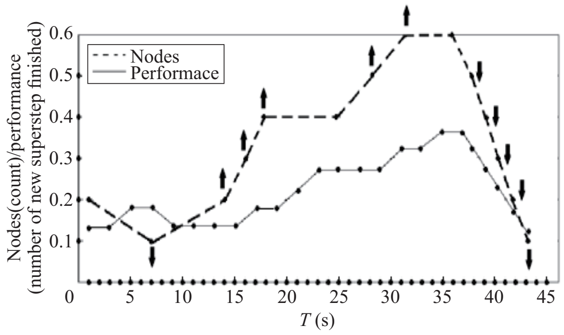


Fig. 7 T-nodes and performance chart.

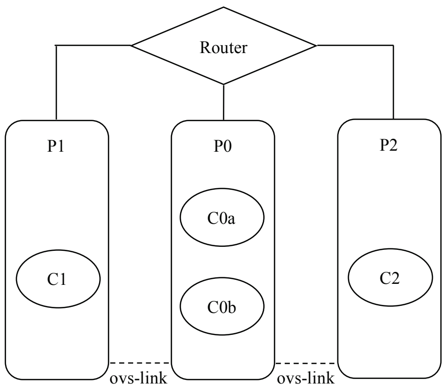


Fig. 8 Topology of testing environment.

P0, P1, and P2 represent physical hosts, and C0a, C0b, C1, and C2 represent containers. The testing result in Table 2 shows that communication through ovs-bridge devices will be much slower than through pure physical devices. The throughput between two physical machines can be three times faster than that between two LXCs on two physical machines. Throughput between two containers on the same physical machine is rapid but also slower than the physical method. This finding implies that the network bridge provided by the current OpenVSwitch based on traditional hardware is not as efficient as expected, which should be optimized by VLAN or newly proposed SDN[13] optimized chip.

6.6 Evaluation: Live migration

The migration of containers would be much faster than traditional VMs in terms of the amount of data transmission of file-system and run-time status. For example, files in file-system are divided into multiple Read-Only layers and one Read-Write diff layer based on Union-FS. Therefore, all Read-Only layers can be synchronized on each machine in advance, leaving the diff layer with size that depends on files changed by run-time applications needed for migration; this feature would reduce the time in transmitting the file block such as VMDK14.

For live migration of run-time states of a container such as memory pages, CPU registers, and network buffers, we use CRIU to conduct a checkpoint of them and recover live states on another machine. With the help of CRIU, LXC provides "lxc-checkpoint" command for live migration, which is supported by OS system: Ubuntu 15.10 (Wily), Linux Kernal v4.2, CRIU v1.72, and LXC v1.1.4.

Compared with live migration of VMs, the total amount of run-time status in a container mainly depends on the actual memory usage apart from memory fragments, which is much efficient than whole data block transmission of VM memory that is based on loop copying of dirty pages within a long time. Table 3 presents the time cost and data size for live migration of a container based on CRIU.

"C 1M int" means the total memory cost of services inside a container is equal to a sequential 1M-size integer array (such as "new int[1<<20]"). Similarly, "N 10M int" means the total memory cost is equal to a non-sequential 10M-size integer linked list (such as a 10M-length linked list created by C++ structure like "struct node node* next; "). T(chk) is the time CRIU conducts a checkpoint for a run-time container, T(rec) is the time CRIU performs live recovery of a container, and S(data) is the total size of intermediate status of a run-time container to copy from one host to another.

7 Conclusion and Future Work

This paper presents the concept of the ClaaS model for general purpose and a simple implementation of the Docklet system, which has the advantages of both OpenStack and Mesos, providing scalability, compatibility, and software definition for many application frameworks.

ClaaS is an effective and easy-to-use model for sharing a physical cluster among several frameworks. It extremely simplifies the process of deployment and configuration of various applications (especially for distributed applications), which can be shared by multiple users and rebuilt at any time by software protocol easily. The Docklet platform can generate equivalent Vclusters independently from fixed hardware topology, with some mechanisms specifically designed to make containers function better in a network environment. Several policies can be used by users to handle migration and booting priority of nodes. The final evaluation results show that the scaling and elastic effect of Vcluster is good.

Future work includes optimization of load balancing for fine-grained nodes on each machine and smart

Table 2 Data transmission throughput between nodes.

Source	Destination	Throughput (MB/s)
P0	P0	924
P0	P1	660
P1	P2	662
P0	C0a	398
P0	C1	364
C0a	C0a	772
C0a	C0b	395
C0a	C1	227
C1	C2	231

Table 3 Relations between run-time containers using different memories and their time costs and intermediate data size.

Memory usage	T(chk) (s)	T(rec) (s)	S(data) (Byte)
0M int	0.199	0.116	5.70×10^6
C 1M int	0.275	0.131	9.80×10^6
C 10M int	0.623	0.158	4.49×10^7
C 100M int	3.906	0.274	3.96×10^8
C 200M int	8.308	0.413	7.87×10^8
N 1M int	0.535	0.149	3.71×10^7
N 10M int	3.263	0.227	3.18×10^8
N 100M int	27.864	24.202	3.13×10^9

control of resources used in both horizontal scaling and vertical scaling.

Acknowledgment

This work was supported by the National Science and Technology Major Project (No. 2016YFB-1000105); the National Natural Science Foundation of China (No. 61272154); and the Science Fund for Creative Research Groups of China (No. 61421091).

References

[1] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, Mesos: A platform for fine-grained resource sharing in the data center, in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, 2011, pp. 295–308.

[2] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg, in *Proceedings of the Tenth European Conference on Computer Systems*, 2015, p. 18.

[3] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing, in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI14)*, 2014, pp. 285–300.

[4] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters, in *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013, pp. 351–364.

[5] LXC overview document, http://lxc.sourceforge.net/lxc.html, 2016.

[6] D. Bernstein, Containers and cloud: From lxc to docker to kubernetes, *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[7] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, An updated performance comparison of virtual machines and linux containers, in *2015 IEEE International Symposium on Performance Analysis of Systems and Software*, 2015, pp. 171–172.

[8] F. Doelitzscher, M. Held, C. Reich, and A. Sulistio, Viteraas: Virtual cluster as a service, in *2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, 2011, pp. 652–657.

[9] M. Brock and A. Goscinski, A technology to expose a cluster as a service in a cloud, in *Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing*, Australian Computer Society, 2010, pp. 3–12.

[10] OpenStack, NASA and Rackspace, http://docs.openstack.org, 2016.

[11] D. Quigley, J. Sipek, C. P. Wright, and E. Zadok, Unionfs: User and community-oriented development of a unification file system, in *Proceedings of the 2006 Linux Symposium*, 2006, pp. 349–362.

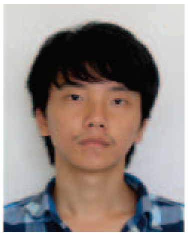
[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, OpenFlow: Enabling innovation in campus net, *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[13] Open Networking Foundation, Software-defined networking: The new norm for networks, https://www.opennetworking.org/ja/sdn-videos-bc-2/46-sdn-resources/sdn-library/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, 2016.

Donggang Cao is currently an associate professor at Software Institute, School of Electronics Engineering and Computer Science, Peking University. He received the PhD degree from Peking University in 2004. His research interests include system software, parallel and distributed computing, etc.

Peidong Liu is currently a master student at Software Engineering Institute, Peking University, China. He received the bachelor degree from Peking University in 2015. His research interests include big data, system software, parallel and distributed computing, etc.

Wei Cui is currently an engineer at EMC Corporation. He received the master degree from Peking University in 2016. His research interests include big data, system software, parallel and distributed computing, etc.



Yehong Zhong is currently a master student at Software Engineering Institute, Peking University, China. He received the bachelor degree from Peking University in 2016. His research interests include big data, system software, parallel and distributed computing, etc.

Bo An is currently a PhD candidate at Software Engineering Institute, Peking University, China. He received the bachelor degree from Peking University in 2014. His research interests include big data, system software, parallel and distributed computing, etc.