A Virtual Machine Data Communication Mechanism on Openstack

Jie Chen[1,2(✉)], Saihong Xu[1], Haiyang Zhang[1], and Zunliang Wang[1,2]

[1] School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, People's Republic of China
chenjie52388@163.com

[2] Science and Technology on Information Transmission and Dissemination in Communication Networks Laboratory, Shijiazhuang, China

Abstract. This article analyzed the advantages and disadvantages of current data communication mechanism among multiple virtual machines. Combining the characteristics of the cloud platform openstack, this paper puts forward a method to make the communication of virtual machines more efficient by using para-virtualization. Through the shared memory we can break down the communication barriers among the virtual machines and reduce the number of data copy times in the process of data transferring. Experiments show that the communication efficiency of multiple virtual machines gets higher.

Keywords: Communication of multiple virtual machines · Para-virtualization · Shared memory · KVM · Virtio

1 Introduction

Nowadays, the service provider around the world is adopting virtualization and cloud computing technology to deploy and deliver virtual network services, namely network function virtualization (NFV). Deploying services in virtual machines makes the deployment cost less and efficiency higher. And how to communicate among virtual machines more efficiently is becoming the focus of attention.

The continuous development of virtualization technology makes the dependencies of network function and its hardware eliminated. Just as the virtual network function (VNF) of NFV, it decouples the software and hardware. The most direct benefit is that we can create multiple independent virtual machines on the same set of physical hardware. These virtual machines can provide different functions. At present, the most popular and mature virtualization technologies include KVM, Zen, etc.

At the same time, the cloud computing has a rapid development in recent years. Cloud computing has become one of the hottest technology. It will combine a variety of software and hardware to constitute a powerful computing platform which can provide customers with various services through the network. It includes infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS). We can manage and schedule the computing resources using these technologies. NFV for the usage of cloud computing mainly concentrates in the IaaS layer. Here we consider openstack. We can use it to deploy virtualized environment quickly, and can create multiple interconnected virtual machines, namely VNF. And users can quickly deploy applications on the virtual servers.

With the mature of virtualization and cloud computing technology, the virtual machine is widely used today. Enterprise uses a set of physical resources to create many virtual machines, not only for their own use, but also providing services to others, such as economical cloud hosting now. These cloud hosting its essence is a series of virtual machines. This paper mainly studies the data communication mechanism of virtual machines based on openstack.

2 The Communication Scenarios

2.1 Communication Scenarios

Network function virtualization (NFV) has an application scenario, which is IP Multimedia System (IMS). The IMS system contains many entities. For the sake of simplicity, here we only consider the MRF. MRF is mainly completed for many calls and multimedia conferencing. MRF consists of MRFC (Multimedia Resource Function Controller) and MRFP (Multimedia Resource Function Processor). They complete the media flow control and load function respectively. In some improvements of IMS system such as NCSP [1], there will be an extra MRMS (multimedia resource management system) entity. The MRMS implements the management function of the whole system as shown in Fig. 1.

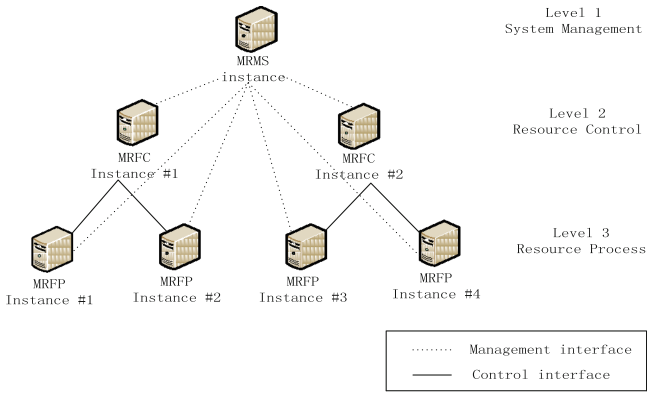

Fig. 1. MRF

2.2 Communication Requirements

In the multimedia resource management subsystem, MRMS is the manager of the MRF. It has two big functions: First, it provides query and modify function for the subsystem's software information of MRF; Followed by operation maintenance function, it is the main function of MRMS subsystem, providing query function, configuration function, operation function, authentication and so on.

The function mentioned above involves a large amount of data communication. In addition, in 3 GPP's IMS architecture, for MRFC and MRFP, MRFC is responsible for interacting with call session control server, receiving the control requirements for the media dealing from the application layer. And then process the received control requirements. Call MRFP to deal with media stream. These media stream includes audio, video, and signal data. So the data to be interacted is very large. Finally MRFP gives treatment results back to MRFC. We can see the process between MRFC and MRFP existing large amounts of data communication.

In such a scenario, we create many virtual machines to act as MRMS, MRFC and MRFP based on the same set of hardware. Facing a large amount of data communication in these virtual machines, the problem we need to solve is to make the communication as efficient as possible.

3 The Methods of Communication

Nowadays, there is many ways of data communication among virtual machines. Here we list some and compare them.

3.1 Pipeline

Pipeline is the most basic method of IPC. Usually a pipeline created by a process, and then the process calls fork to create a child process. Then the father process and the son process can use the pipeline to communicate. After calling pipe to create a pipeline, it will receive two file descriptor. One for writing, another is used to read. Due to the anonymous pipeline is half duplex, if we want to undertake two-way communication, we will have to build another pipeline.

3.2 Fifo

Also known as the named pipeline, FIFO is represented by a special file. It is longer limited to kinship between processes. For the communication among many virtual machines, the approach taken is to create a public FIFO to upload the data. And then to the public FIFO there is a monitoring process, the monitoring process takes the data from the pipeline. For every virtual machine in need to establish a dedicated FIFO, the monitoring process use it to transfer the data obtained from the public pipeline.

3.3 Message Queue

The message queue method uses the communication mechanism of processes. Through the public system message queue to exchange data among two or more processes, this way realizes the reliable message receiving and dispatching mechanism. Message queue technology is a kind of technology for the exchange of information among distributed applications. Message queue can reside in memory or on disk. Queue stores messages until they are read by applications. Through the message queue, the applications can perform independently. They don't need to know the position of each other.

3.4 Shared Memory Channel

In a large amount of data communication scenario, many people use shared memory way, which has the advantage of high efficiency. Shared memory is part of the physical memory shared by multiple processes. It is the fastest method for processes to share data. A process writes the data to the shared memory region, then all other processes who shared this memory area can soon see its contents. They exchange data through the shared memory region.

In all communication methods above, using pipeline is the worst. Because its half duplex communication mode is not convenient. Bidirectional communication need to build two pipeline which takes more resources; Using FIFO scheme with fewer pipelines is its advantage, but there is one drawback that it needs to maintain a mapping table, to record one-to-one correspondence relationship between the user ID and a dedicated FIFO. So the resource consumption is also big; Compared with the former two, message queue greatly simplifies the program logic relationship and dynamic mapping table is omitted which reduced resource consumption. But there is also a problem that the message queue is system level resources. It will not automatically shut down and the message will not disappear. Remaining messages in the queue accumulated to a certain degree will cause the performance loss of the system; The shared memory way, in terms of resource consumption, communication efficiency and flexibility are better than the front several ways. Because the virtual machines are based on openstack, the virtual machines ard based on the same set of physical hardware. Through the way of the Shared memory communication between them is possible, so the proposed method has also adopted the shared memory way. But the most primitive method of shared memory channel can't solve the problem of data communication among multiple virtual machines. It is that it can't solve the application scenario of multiple virtual machines communication problems. Because it needs to maintain a large number of shared memory region, the consumption of resources is too big. On the basis of openstack using the original shared memory model doesn't work. So this paper proposes a method of communication among multiple virtual machines on openstack, it can allocate shared memory for target virtual machine to read or write, and we can map dynamically in the building of a link. That can reduce a large amount of shared memory resource consumption when maintaining the system.

4 Design and Implementation

Openstack supports almost all kinds of hypervisors. The openstack we considered here is based on KVM, which is usually the default hypervisor. So the virtual machines created by openstack are a series of KVM virtual machines. However, the inter-domain communication of virtual machines based on KVM is cumbersome in the processing procedure. It needs to copy data repeatedly which impacts on the performances greatly.

Most of the behaviors of transmitting and receiving data packets in the network I/O process of a virtual machine client are simulated by the QEMU program in the host user mode, which equivalents to transfer data between host processes. So we can use the characteristics of host's Linux system to design and implement a communication mode based on shared memory. In order to implement the communication mode of shared memory among the virtual machines, we need para-virtualization programming interface virtio to implement equipment to offer direct network access based on shared memory. This method can reduce data copy times in inter-domain communication and improve communication efficiency.

4.1 Shared Memory

The above-mentioned virtual machines are created on the same physical machine. So we can use shared memory approach to implement the data communication among these virtual machines. The advantage of shared memory is high efficiency, suitable for the usage in a large number of data communication.

The method adopted in this paper is mapping the shared memory region when multiple virtual machines need inter-domain communication. The advantage of this method is that we can allocate a shared memory area and each virtual machine can dynamically map to it during the establishment of the link. So there is no need to maintain a large number of shared memory areas. This solves the synchronization issue of a large amount of data among multiple virtual machines. For the scenarios previously mentioned, this is a good solution. Greatly improve the efficiency of data transmission among virtual machines, and the response becomes faster.

For this shared memory architecture, it can be put into two parts in design. First, in order for scalability, each time creating a virtual machine, when need to communicate with other virtual machines, allocate a block of memory mapped into its address space, all virtual machines each correspond to a block of memory. Make these blocks of memory to form a circular linked list. When given two virtual machines need to synchronize data, we can find the corresponding block belongs to the peer virtual machine according to the circular linked list, and inform this virtual machine's memory address, both at the same time can write each other's virtual machine information to the corresponding memory of the peer side, which can greatly enhance the efficiency of data synchronization; Second, apply additional block of memory, so that all virtual machines are mapped into this block, this block of memory's size varies according to different scenarios. For the global data that all virtual machines need to synchronize, we can synchronize through this shared memory. Model is shown in Fig. 2.

4.1.1 The Communication Model

This section describes how to use the shared memory model in the whole process of communication. Based on the same set of physical hardware, the communication among virtual machines can be called domain communication. The domain communication refers to the communication happens on some clients which are on the same physical machine. The communication model is divided into two parts: the VM agent and Scanner agent. As is shown in Fig. 3.

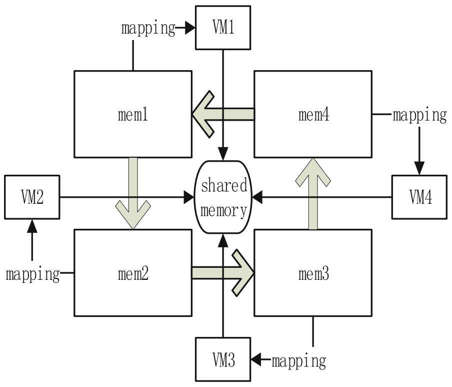

Fig. 2. Shared memory model

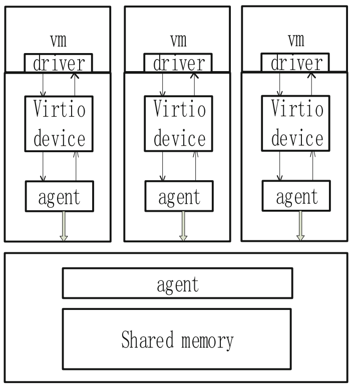

Fig. 3. Communication model

The agent at the bottom, its main job is to get all virtual machines' information using the API of openstack, such as virtual machine's IP and Mac address, so that the virtual machine can get the target machine's information when in need. The communication model among the virtual machines take the page as the basic unit of the Shared memory, through a series of pages constituting the block of memory to share [2]. In the communication model, most of the work is done by client virtual machines' agent. Its main task is to:

(1) Get the mac address of target virtual machine
(2) Query the underlying agent to get the id of target virtual machine
(3) Confirm whether the target virtual machine on the same physical machine
(4) Establish the shared memory communication.

4.1.2 The Process of Creating Shared Memory

Then we will list all kinds of communication scenarios among vm1, vm2 and vm3:

(1) Point-to-point Communication (vm1, vm2). According to the aforementioned shared memory structure and communication model, first vm1 agent send a query request to share layer agent to detect whether vm1 and vm2 is in the same physical machine. At the same time, the bottom agent tell vm2 that vm1 want to communicate with it with vm1's information appended. Then according to the information to allocate a piece of memory mem1 and mem2 respectively. Then vm1 write data to mem2 and vm2 write data to mem1. So can greatly improve the communication efficiency.
(2) The Communication of Multiple Virtual Machines (vm1,vm2 and vm3). Similar to the above process, the process of vm1 launched communication is as follows. First, vm1 through the agent to take a request of communication, then sharing layer agent determines the communication belongs to multiple virtual machines communication. Then determine whether all virtual machines are located in the same physical machine. If it is, apply for a large block of memory, again according to the relevant information of each virtual machine, the memory mapping to their corresponding

process address space. And then inform each virtual machine the environment is ready, you can communicate with each other.

(3) The Comparison with Other Communication Model. First, the communication method this paper presents compared with other not shared memory communication mode, greatly improves the efficiency of the communication among the virtual machines, according to the actual situation to use the characteristics of the virtual machines on openstack; Second, compared with other Shared memory communication mode, this model not only solves the point-to-point communication between two virtual machines, it can also act as a more effective communication mode among multiple virtual machines, hoping to provide certain reference for a more perfect and efficient communication mechanism among the virtual machines.

4.2 Full Virtualization

Linux full virtualization solution contains hardware virtualization technology mainly for the extension of the X86 architecture. Full virtualization runs faster than hardware emulation, but performances less than bare machine, because the hypervisor takes up some resources. The great advantage of full virtualization is the operating system without any modification. The only limitation is that the operating system must be able to support the underlying hardware. From the point of the full virtualization, the whole process of using KVM is that the bottom of the hardware which supports x86 virtualization extensions (such as AMD and Intel VT- the SVM technology) loads the KVM kernel module of the Linux kernel as a virtual machine manager. Then use KVM tools to load the client operating system and use QEMU to simulate virtual equipment and process the I/O requests.

After intercepting I/O instructions, KVM will redirect and simulate the I/O instructions. These operations are implemented by QEMU, a slightly modified software. QEMU simulates card, memory, bus and other hardware components, supplying a complete I/O model for the clients. Managing the memory by maintaining the mapping between the client's physical address and the host's physical address. In addition, QEMU will also provide virtual device interface for each client, using these to process the received requests, the specific process not described in detail here. I have to say, although the whole KVM virtualization architecture is simple and it takes full advantage of the performance benefits of Linux system, but in terms of virtual network implementation, it is far too complicated, especially low in performance when going on domain communication among virtual machines.

4.3 The Interface Virtio

KVM supports full virtualization and para-virtualization technology. Using full virtualization technology, the domain communication efficiency among virtual machines performs poorly. So it is hoped that through the para-virtualization mechanism to achieve better I/O performance. Para-virtualization similar to full virtualization is a popular virtualization technology. Here we use it to implement shared memory mechanism. It uses the Hypervisor (virtual machine management program) to share and access the underlying hardware, and the guest operating system has integrated the code of virtualization. Using para-virtualization mechanism KVM can achieve better I/O performance. At present, a general efficient I/O para-virtualization mechanism has been adopted by KVM, implemented by para-virtualization programming interface virtio. Virtio is a para-virtualization model suitable for multi-platform. The architecture of virtio is illustrated in Fig. 4.

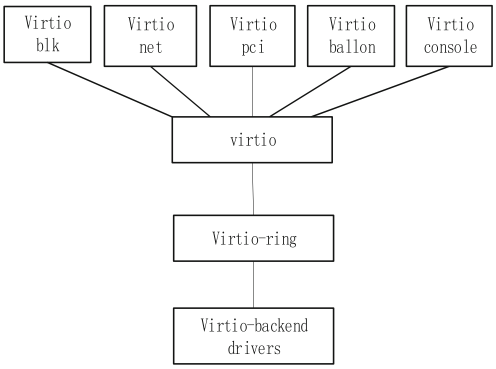

Fig. 4. Architecture of virtio

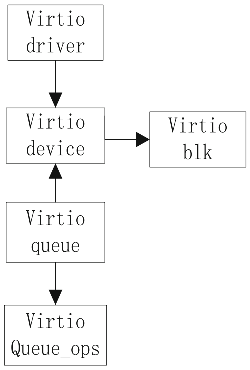

Fig. 5. Architecture

Virtio drivers are designed to be similar to the universal drivers and operating structures, as shown in Fig. 5. Virtio para-virtualization driver is divided into front driver and backend driver. It defines two layers to support the communication from the guest operating system to the hypervisor. The virtio_driver is in the top and it represents front-end drivers in the guest operating system. The devices matched with the drivers are encapsulated by the virtio_device in the guest operating system. Each virtio driver corresponds with back-end virtio device and back-end equipment is responsible for handling the requests from the interaction of front and end. Virtio_device get functions of receiving, dispatching and configuration by using the data buffer of virtio_queue object. Finally, each virtio_queue object references the virtqueue_ops object and the latter defines and deals with the underlying queue operations of the hypervisor drivers.

In the present of KVM virtual machines, the corresponding virtual bus, virtual block devices and virtual network devices have been implemented using virtio. According to the test results, the network throughput is improved about three times.

4.4 Implementing Virtual Devices

In view of the shared memory method mentioned above, the device model is realized using the programming interface virtio. The front and back-end I/O use 3.5 virtual queue to be responsible for receiving, sending the packages and the acess to control area. Besides, there is a structure, which is used to save the data descriptors taken from the virtio_queue. Each virtual machine uses the inter domain channel to communicate, which uses the interfaces of descriptors buffer to communicate with the virtual device virtio_inter. Data processing flow is to add buffer descriptors to the virtqueue or remove from it to transfer the requests. When the data packet is written to the shared memory channel, the client will call virto_inter_receive method to receive the data. Calling process is shown in Fig. 6.

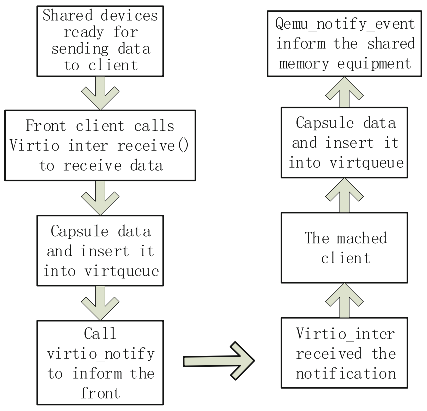
Shared devices ready for sending data to client
Front client calls Virtio_inter_receive() to receive data
Capsule data and insert it into virtqueue
Call virtio_notify to inform the front
Virtio_inter received the notification
The mached client
Capsule data and insert it into virtqueue
Qemu_notify_event inform the shared memory equipment

Fig. 6. Process

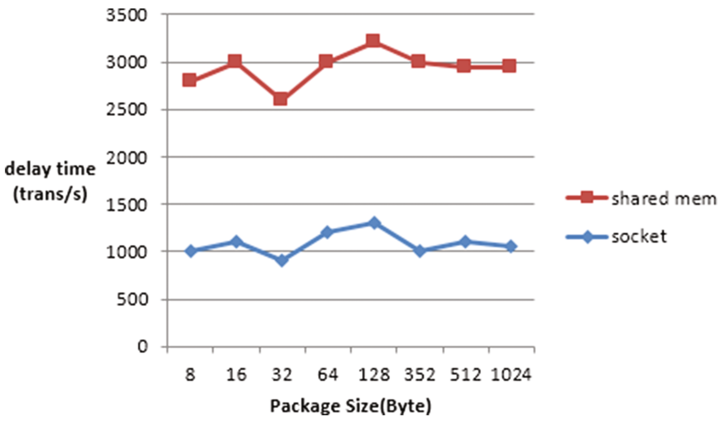

Fig. 7. Results

5 The Performance Analysis

In this section, we test the communication efficiency among multiple virtual machines, and analyze the test results with system ubuntu 14.10, I5 processor and Hz 2.53 G. The main contrast is the two cases: First, the throughput and transmission delay of communication of multiple virtual machines through the socket and other conventional methods to transfer a large amount of data; Second, get the communication results of multiple virtual machines through the shared memory method. The results are shown in Fig. 7 above.

From the Figure we can get that the way of shared memory reduce a large amount of data copy and transmission times, which can reduce the response time delay. As shown in the figure, the horizontal coordinates represent the size of the data packet and the unit is Byte. The longitudinal coordinates use receiving and transmission rate on behalf of the corresponding delay performance and the unit is tranction/s, that is the number of sending times per second. From the graph, we can see that the way of shared memory does make the efficiency of data communication higher and the delay lower.

6 Conclusion

This paper proposes a method showing virtual machines on openstack how to communicate with the shared memory. It makes use of paravirtualized I/O model virtio to simulate virtual network devices and corresponding client's front-end driver, utilizing the approach of shared memory to transfer data. The results of the test show that this method can improve the efficiency of data synchronization among virtual machines. The problems of the scenarios mentioned above can be solved to a large extent.

Acknowledgement. This work was supported by the open project of Science and Technology on Information Transmission and Dissemination in Communication Networks Laboratory (ITD-U14002/KX142600009).

References

1. Chen C.L.: Improvements in Functionality and Deployment Structure of MRF in IMS (2012)
2. Shengge, D.I.N.G., Ruhui, M.A., Alei, L.I.A.N.G., Haibing, G.U.A.N: Optimization for Inter-VMs Communication on KVM with Para-Virtualized I/O Model, September 2011
3. Diakhate, F., Perache, M., Namyst, R., Jourdren, H.: Efficient shared memory message passing for inter-VM communications. In: César, E., Alexander, M., Streit, A., Träff, J.L., Cérin, C., Knüpfer, A., Kranzlmüller, D., Jha, S. (eds.) Euro-Par 2008 Workshops - Parallel Processing. LNCS, vol. 5415, pp. 53–62. Springer, Heidelberg (2009)
4. Russell, R.: Virtio:towards a de–.factio standard for virtual I/O devices. J. ACM SIGOPS Oper. Syst. Rev. **42**(5), 95–103 (2008)
5. Kivity A., Kamay Y.L., Laor, D. et al.: KVM: the Linux virtual machine monitor. In: Proceedings of the Linux Symposium, Ottawa, Ontario, pp. 225–230 (2007)
6. Shengzhao, L., Qinfen, H., Limin, X. et al.: Optimizing network virtualization i kernel—based virtual machine. In: Procedings of the 1st IEEE Internal Conference on Information Science and Engineering (ICISE 2009), pp. 282–285. IEEE Computer Society, Washington DC (2009)
7. Chert, G.I., Lai, T.H.: Constructing parallel paths between two subcubes. IEEE Trans. Comput. **41**(1), 118–123 (1992)

8. Stankovie, J.A.: Implication of classical heduling results for real-time systems. IEEE Comput. **28**(6), 16–25 (1995)
9. Bellard, F.: QEMU, a fast and portable dynamic translator. In: Proceedings of the 2005 USENIX Annual Technical Conference (USENIX 2005), Marriott Anaheimm, April 2005. USENIX Association, Berkeley, CA, USA (2005)