Virtual Machine Co-migration *

Saurabh Kadekodi
Northwestern University
Evanston,IL
saurabhkadekodi2013
@u.northwestern.edu

Chao Shi
Northwestern University
Evanston,IL
chaoshi2012
@u.northwestern.edu

Qingyuan Zhang
Northwestern University
Evanston,IL
zqy@u.northwestern.edu

ABSTRACT

The migration of virtual machine from one physical node to another is a common practice in the modern data center. In this paper, we developed a framework which is able to do virtual machine co-migration. Different from the classical single source to single destination migration, Co-migration will migrate and instantiate a group of virtual machines on the destination physical nodes. We test our framework using Palacios, a OS independent virtual machine monitor (VMM). The experiment results show that our framework can guarantee the integrity of virtual machine pages as well as satisfactory throughput.

Categories and Subject Descriptors

D.4.2 [**Storage Management**]: Memory, Networking

General Terms

Design

Keywords

Virtual machine, Live and dead migration, Co-migration, Reliable UDP

1. INTRODUCTION

Co-migration is the process of collaboratively migrating a group of virtual machines to different destination physical nodes. The source virtual machines can be deployed on more than one physical nodes and managed by different instances of virtual machine monitor. The goal of co-migration is to migrate them as a whole group to the destination with high throughput and little delay. Co-migration can either be live or dead. The application service running on of the virtual machine will not halt while the migration is in progress, and on the contrary, in dead migration, the the memory page will be snapshotted before migration. For simplicity reasons, we are only implementing the dead migration version.

One of the advantages of co-migration is that it utilize the shared memory pages among different virtual machines in an optimized way. In the real world, it is more common to migrate a service which is provided by a group of virtual machines having large portion of duplicated memory contents. By addressing the contents in the distributed hash table, the co-migration method can provide high efficiency by reducing the number of pages being transmitted over the network. Secondly, the co-migration will provide load balancing and provide high throughput by coordinating among source physical nodes. For example, if one of the VMMs is experiencing trouble sending pages, the other VMMs can take over its job. The nature of page content sharing and parallelism both improve the efficiency.

In our project, we implement our framework for doing virtual machine co-migration on the platform of Palacios, which is a OS independent virtual machine monitor. Palacios can be compiled into a linux kernel module and provides many useful interfaces to the linux library. Before our development, palacios only provides some tools which can save the state of the virtual machine to a remote place across the network. Now, the user can co-migrate a group of virtual machines by giving the co-migration mapping file as the input. The file indicates a full list of 6-dimensional tuples in the format of {src_vmm_ip, src_vm_name, src_page_num, dst_vmm_ip, dst_vm_name, dst_page_num }.

To obtain the co-migration mapping, we have to utilise CONCORD, a system which is based on distributed hash table and is able to track the memory pages. Given the hash value of the memory page, it will inform the virtual machine monitor how the memory pages are duplicated among different virtual machines.

When transferring the memory stream, we are using UDP instead of TCP. The major reason is that UDP is little in handshake and out-of-order retransmission overhead and thus more scalable. We add some additional features to provide a connectionless transmission mechanism which guarantee data integrity but not correct order. The major reason is that page arrival order is not important and this sacrifice can greatly reduce the number of retransmissions.

The following of the paper is organized as follows. Part 2 will briefly introduce the hot topics in the research of virtual machine migration. Part 3 will discuss our design idea and implementation details. Part 4 will show some of our experimental results and part 5 will be examining the future work.

*Summary paper for the quarter long project of EECS441 Resource Virtualization

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WOODSTOCK 'EECS441 Resource Virtualization
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

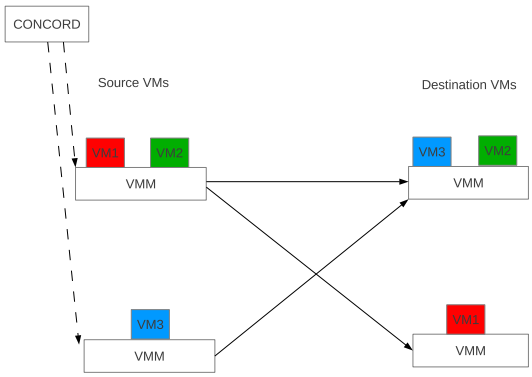

Figure 1: Architecture for Virtual Machine Co-migration

2. RELATED WORK

A number of mechanisms have been proposed to solve the problem of virtual machine migration and instantiation. Most of the focuses was on transferring memory pages from a single source to a single destination over a local-area of wide-area network. Much work has been done on the live migration and many among them [1] [2] are using the assistance from a shared storage system for virtual machine memory pages. VMFlock [3] is the working on co-migration of a cluster of virtual machines and our framework is similar to that in some aspects. For example, VMFlock is able to chunk the memory of virtual machines and address them using it hash value, which is somehow sharing the idea with CONCORD. Besides, VMFlock is able to exploits the similarity of the pages to be transmitted and thus reduce the packets over the wire. One major difference from VMFlock is that we are also considering the efficiency of data transmission on the network, which becomes a problem when the size of the co-migration virtual machine group grows up.

3. SYSTEM DESIGN AND IMPLEMENTATION

Figure 1 briefly summarises the architecture of our framework. There are multiple physical nodes for the source group and each of them will run a separate virtual machine monitor. On each virtual machine monitor, there can be multiple virtual machines running on top. CONCORD is a separate system which is based on distributed hash table for storing the memory page hash. By contacting CONCORD with the input of the memory page hash value, the virtual machine monitor is able to know the exact location of the duplicated memory pages.

The virtual machine monitor has full control over the physical memory space of the virtual machines. The physical memory space of virtual machine is continuously mapped onto the a physical address of the host machine, which saves us the trouble of locating a specific page content on the host. The virtual machine monitor will do the page transmission on behalf of the virtual machines. From the perspective of the virtual machine monitor, the page content is merely a byte stream starting at the host virtual address translated from the host physical address called guest memory base pointer. We simply implemented a UDP interface which is able to deal with reliable byte stream transmission in the first half of the quarter.

3.1 Reliable UDP

As mentioned before, one of the reasons we are using UDP instead of TCP is scalability and little overhead.UDP is a best-effort connectionless transport layer protocol which only provides us a checksum to detect whether the datagram received is corrupted or not. If the datagram is corrupted or lost, there is no retransmission process, and there is no mechanisms to guarantee that the datagrams arrive in the right order.

For our project, the integrity of the transmitted memory pages is critical because otherwise, one will not be able to reboot it afterwards, and therefore there have to be some mechanisms to guarantee that the correct page contents finally arrives. The right order, however is less important for our project, this is because each memory page encapsulated into the packet is tagged with the metadata containing the page number. The receive does not have to access the memory sequentially and can write one page into memory even if it comes out of order.

3.1.1 Design Idea

We implement our extended version of UDP protocol in application layer rather than directly change the code in the transport layer. Different from the traditional UDP, the receiver will send out an acknowledgement message back to the sender encapsulated in the normal UDP datagram when the previous data has been successfully received with no corruption. The sender will resend the packet if it does not receive the packet within a certain time frame. The timeout value for now is 5 seconds. Besides, there is a limit to the number of retransmission, which is set to be no larger than 5. This method is considered much simpler to be implemented.

The data to be transmitted is tagged with some format of metadata, which is not trivial. The destination ip address and port number is self-explanatory. The buf is used to convey the byte in the page, and the MAX_PACKAGE_LENGTH denotes the maximum number of bytes in a page. In the real world case, the memory pages are in the size of 4KB, but to guarantee that the frame size does not exceed the value of MTU, we split one page into four parts and encapsulate them in four packets. MAX_PACKAGE_LENGTH is set to be 1024 in our scenario. len, type and package_number are some control information values which are used to assure integrity. The guest number belongs to one of the virtual machines running on top of the destination VMM and together with the page_number is used to write the byte stream to a specific memory page inside that virtual machine. The data is sent together with the metadata in transmission as a whole packet.

```
struct pkg_struct
  {
      unsigned int dest_ip;
      int dest_port;
      char buf[MAX_PACKAGE_LENGTH];
      int len;
      int type;
      long package_amount;
      int guest_number;
      long page_number;
```

```
}
```

While the structure for the acknowledgement packet is much simpler.

```
struct ack_pkg
  {
      int type;
      long seq_id;
  }
```

Similar to the selective repetition algorithm, the receiver is able to selectively acknowledge the arriving packets. The most important difference is that the receiver does not need to buffer the out-of-order packets, instead it can directly surrender the bytes got from the wire to the party interested even though the packet does not have the next sequence identifier expected. For the sender, there still exists a peding_list containing all the packets which are sent out and not ACKed. The packet is removed from the list when the corresponding ACK arrives and resent over the wire again if timeout. For the receiver party, there is no such a list put the packets back in order, namely with window size being 1.

3.1.2 Thread Model

The sender and receiver will be both implemented with multi-thread concepts. We use multi-thread in order to avoid blocking I/O. The sender will basically have two thread. One thread is for receiving the inbound ACKs from the other side. Another thread is for timing and resending. The first thread for ACK receiving will listen on a well known port number and capture all the ACKs. It will then traverse the pending list and remove the ACKed packet from it. It will make modifications for the window if needed. It will also reset the timer if there are still pending packets in the list. For the timout thread, it will sleep for the interval of the timeout value each round, and when it wake up, it will retransmit the first packet in the pending list.

The receiver side is pretty straightforward, it only contains one thread for receiving all inbound packets and ACK them if the checksum is correct.

3.1.3 Implementation

As is seen later, the major code execution happens in the palacios itself, which is the virtual machine monitor. The codebase tell us it is really hard to implement your code inside palacios. The reason is you cannot use any user space library functions, and besides you have to use the functions equivalent to linux kernel functions from palacios itself. To solve this problem, we are implementing our reliable udp in the space of linux_module where we have all access to linux kernel functions. The when the code in palacios want to use that service, it will use the socket hook in vmm_socket.c to initiate those function calls.

3.2 Sending Process

For the sending process, we are actually calling ioctl on the VMM device itself rather than a specific virtual machine this is because there are multiple virtual machines to be migrated and calling ioctl on each one is cumbersome. Another reason is that VMM has a good view of the mapping of the guest physical address to the host physical address.

The VMM will be able to get the data structure for each virtual machine running on top of it. The host physical address where the physical guest base memory is mapped to (HPA) can be accessed through the structure. The host physical address for a particular page base can be calculated by adding the the product of page number and 1024 to HPA. After that, there is a macro translating that host physical address into host virtual address. We can use that host virtual address as the starting pointer for memory page copying and iteratively reading for 1024 bytes.

The byte stream read is put to the interface for reliable udp transmission.

3.3 Receiving Process

The process is similar to sending process. The receiver will read from the packet the memory page content as well as the metadata. It will pinpoint the virtual machine to write memory into. Like sending process, it will calculate the host virtual address for the HPA where that particular page is mapped onto. Then it will write this byte stream into the host virtual address. The receiving process will also tell the UDP interface about how many bytes are successfully written, and ask the UDP receiver to pass the data again if there is write-to-memory failures.

3.4 Scatter/Gather Map

Each line for the scatter/gather map will contain six tuples. They are:

1. source host IP
2. source vm number
3. source page number
4. destination host IP
5. destination vm number
6. destination page number

Suppose that this file is generated by CONCORD and is shared among all the sending hosts. Each host will open the file and read the records one by one. The IP address not matching itself shall be ignored. Then all the records with the source host IP equal to oneself shall be fabricated into a linked list. This linked list will copy one by one into the kernel memory space. The reason we have to do this is that it is not realistic to read the mapping file in kernel.

3.5 Usage

In order to test our program, we generate a fake scattergather mapping file called fake_map.txt. This map is shared among all senders. In this experiment, we are testing two source hosts for sending and two receiving hosts. We need to run the receivers first.

For every receiving hosts we need to run.

```
v3_co_receive 12000
```

And for every sending node, we will run

```
v3_co_send fake_map.txt 12000
```

4. EXPERIMENT RESULTS

We test our program on Palacios and confirmed that the memory pages can be transmitted with the integrity guaranteed by the memory page checksum. However, the speed of transmission varies a lot in different cases. The best experiment result we have right now is transferring memory pages with the total size of 256MB in 6 seconds.

5. FUTURE WORK

Right now the practical efficiency is not as high as expected as in theory. In the future work, we will try to optimize our implementation to achieve higher throughput. A few ideas about implementation optimization can be using bitmap instead of linked list, and using the Datagram Congest Control Protocol to do congestion control.

In the linux user space utilities, we can actually have more. For example, we will have a v3_co_cancel to cancel a co-migration job. Another issue is try to combine CONCORD with our system so that we do not need to depend on the temporary fake mapping file any more.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we briefly discussed the basic idea of virtual machine co-migration and the advantages of co-migration. Virtual machine co-migration is in theroy exploiting memory-sharing scalable and high efficiency. We looked at our design idea of virtual machine co-migration and particularly discussed in details the implementation of reliable UDP to assist co-migration. We also carried out our experiment and showed that the co-migration implementation is correct.

7. ACKNOWLEDGMENTS

This project is the summary for the EECS441 Resource Virtualization quarter long project with the supervision of Prof Peter Dinda. The authors would like to express their appreciation for the great help from Prof Dinda in the process.

APPENDIX

.1 References

[1] C. Clark, K. Fraser, Steven Hand, et al. Live Migration of Virtual Machines. NSDI 2005.

[2] H. A. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, et al. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. in European Conference on Computer Systems (Eurosys). 2009.

[3] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, âĂIJVmflock: virtual machine co-migration for the cloud,âĂİ in Proceedings of the 20th international symposium on High performance distributed computing, ser. HPDC âĂŹ11. New York, NY, USA: ACM, 2011, pp. 159âĂŞ170.