A simulation of live virtual machine migration

By: Xiaoming Sun BaishanLv Shan Tao

COEN283, Spring 2014

June 10, 2014

Table of Contents

1. Abstract 3
2. Introduction 3
3. Theoretical bases and literature review 5
4. Hypothesis 6
5. Methodology 7
6. Implementation 8
7. Data analysis and discussion 13
8. Conclusions and recommendations 15
9. Bibliography 15
10. Appendices 18

1. Abstract

Virtual machine (VM) migration in Dynamic Resource Management (DRM) is a crucial factor in minimizing the cost of datacenter operation. To facilitate live migration of VM, many approaches has been proposed and tested. However, existing migration techniques are not silent enough for highly utilized clouds due to their latency and network bandwidth consumption. Here, we are proposing a new migration approach, lazy-copy migration, to provide a fast and silent migration during highly utilized clouds operation. The lazy-copy migration transfers only the memory pages accessed at the destination node in the near future by running a VM at the source node and a migrated VM at the destination node simultaneously during the migration. The lazy-copy migration's highly accurate and low-bandwidth memory transfer mechanism enables a fast and silent VM migration to maintain the Service Level Agreement(SLA) of all VMs in the cloud.

2. Introduction

VM is required to be moved from one host to another from time to time. In order to limit the downtime to almost zero, the concept of live migration of VM is defined and studied. Live migration is one of the key selling points for state-of-the-art virtualization technologies. It allows administrators to consolidate system load, perform maintenance, and flexibly reallocate cluster-wide resources on-the-fly. Migration copy includes three parts: memory copy, disk copy, and network connections copy. So far, pre-copy algorithm and post-copy algorithm are the two most well-known and effective approaches for doing live migration. In this paper, our study mainly focuses on the optimization of the live migration of system VMs.

There are two fundamental requirements in VM migration. First, the VM migration should be fast enough to achieve a user-unnoticeable migration. Second, as few as disturbance should be caused by VM migration to achieve a silent migration.

The pre-copy scheme transfers the VM's memory state before transferring the VM context,

whereas the post-copy scheme transfers the VM context first and then transfers its memory state. To avoid exposing the long memory transfer latency, both schemes overlap the execution of the VM context and its memory transfer process as much as possible while consuming the entire network bandwidth. We define such a full-bandwidth consuming migration as a loud migration. Both schemes must stop their execution to transfer memory pages updated during the pre-copy period or to resolve remote page faults caused during the post-copy period. As a result, the pre-copy scheme can cause a very slow migration or even a migration failure for write-intensive workloads, while the post-copy scheme can cause a very slow migration for memory-intensive workloads. Furthermore, both schemes can incur significant performance overhead to other VMs running in the cloud by consuming the entire network bandwidth available. Therefore, existing schemes do not provide a fast and silent migration, which makes them inefficient for highly utilized clouds where the network bandwidth is one of the most performance-critical resources.

In this paper, we are proposing another migration approach, which we named “lazy-copy” to provide a fast and silent migration to some extent. The lazy-copy scheme achieves a fast migration by maximizing the overlapping of VM execution and its memory copy process, and a silent migration by minimizing the network bandwidth usage. The main idea of lazy-copy scheme is to transfer the VM context immediately to the destination node on a migration request, keep running the remaining VM context at the source node to identify memory pages accessed in the near future, and transfer them to the context at the destination node before being requested. We define the VM context at the source node as the “forward context” and the VM context at the destination node as the “following context”.

By arranging the forward context of the VM at the source node to run slightly ahead of the following context at the destination node, the lazy-copy migration can identify performance-critical memory pages accurately and transfer them timely. The forward context of the VM does not affect the functionality of the following context at the destination node. In this way, the lazy-copy migration hides the long memory copy latency, while minimizing the network bandwidth usage. The lazy-copy migration mechanism has better accurate and lower-

bandwidth memory transfer, which can enable a fast and silent VM migration.

We have built a test environment here and designed a set of program to simulate our new lazy-copy algorithm and the post-copy algorithm, and evaluate the performance of the two. Memory copy is the focus of this study, since that is the place where all the page manipulation happens and thus contributes to the operating system understanding the most. Our simulation shows that the lazy-copy approach can reduce the total migration time and the amount of page fault.

3. Theoretical bases and literature review

Live migration of virtual machine has been intensively studied in the past several years. There has been many algorithms proposed, among which pre-copy migration and post-copy migration are with the deepest impact.

Pre-copy migration algorithm has been widely implemented In Xen, KVM, and VMWare[1, 2, 3, 4]. It starts to copy right after the migration request is made while the VM continue to execute on the source node. Iterative copies are used during the pre-copy phase, when the whole memory content is sent in the first round, and newly produced dirty pages are re-sent in the following rounds. A limit of pre-copy is set ahead of time, using either the estimated downtime threshold or the maximum number of iterations, to decide the critical time when source node is suspended and the leftover dirty pages together with the processor state are transmitted to the destination node, where the VM is restarted on. In spite of its prevalence in nowadays VM platforms, pre-copy algorithm still has some disadvantages. The main issue is the time point when to switch from pre-copy phase to down phase. In write-intensive conditions, the system might be forced to stop and transfer even before reaching the predefined limit, resulting in an increased downtime. Furthermore, during the pre-copy phase, too much network bandwidth might be occupied by the repeated copy iterations[5].

Because of the limitation of pre-copy migration during write-intensive workloads, post-copy

migration is developed[6, 7]. It immediately shut down source node for transmission after a migration request is received, and resume the VM execution at the destination node. A background memory copy process is initiated at the same time to transfer the leftover pages and reduce the chance of page fault. Post-copy migration, as a result, minimized the downtime and bounded the number of memory page transfers so that it is more silent compared to pre-copy migration. However, post-copy has disadvantages too. Page fault can still occur when some certain page is not found in the memory of the destination node. If a decrease in spatial and temporal localities leads to an increase in the memory intensity, the overall performance could decrease a lot due to the significant amount of page faults. The limited network bandwidth also prevents the background memory copy process to transfer all the required pages in time.

Our goal is to implement a faster and more silent migration algorithm for live migration of VM. We used post-copy migration as the starting point of our algorithm. Instead of letting source node act completely as a remote disk for destination node memory as in post-copy migration, we resume both source and destination node after the downtime and let source node be the main active node. Whenever some pages are used in source node, they are copied to the destination node and let the destination node run the same process just happened on source node. A background process is still operating to transmit memory content. Synchronization is achieved be the leading of source node and following of destination node. We kept the merit of post-copy algorithm to minimize the downtime and minimized the number of page faults by sending needed pages to destination before they are executed. After these optimizations, the migration should be more silent.

4. Hypothesis

In this paper, we proposed a new algorithm for doing live migration of VM. To overcome the long waiting time due to large amount of page faults in memory-intensive processes, we proposed a method of resuming both source node and destination node after a brief downtime and letting source node be the leading node. Whenever some pages are used, they

are simply copied and transmitted to destination node to facilitate the execution in the latter node. By doing this modification, we expect shorter total migration time, less amount of page faults produced during the post-copy phase, and thus less interference with the on-going processes.

5. Methodology

The main program is migration manager, it is the entry of the program and triggers other operations on Servers and VMs, like create Server, create VM, stop server, and do migration etc.

Physical Servers are simulated by a Java class, it maintains all the memory allocated to VMs as a hypervisor, it also has some variables and arrays to simulate registers and resources.

VMs are simulated by threads. The number of pages allocated to each VM is fixed and page replacement follows a LRU algorithm. Each VM has its own page table and uses LRU algorithm to swap pages. Each VM will run a program to simulate operations on the memory pages. The hypervisor and VMs know the page miss ratio at any time.

When migration happens, VMs are stopped or suspend, memory state, registers' state and others will print out for record, and then copy what we need, after resuming, print the information again. All pages transferred from source to destination will print out as well. Page fault and delay time will also be calculated.

The simulation will be implemented in the Java programming language and compiled with Java compiler under the Linux environment.

This program can set different memory copy schemes, the result will be output into files for further comparison and analysis.

6. Implementation

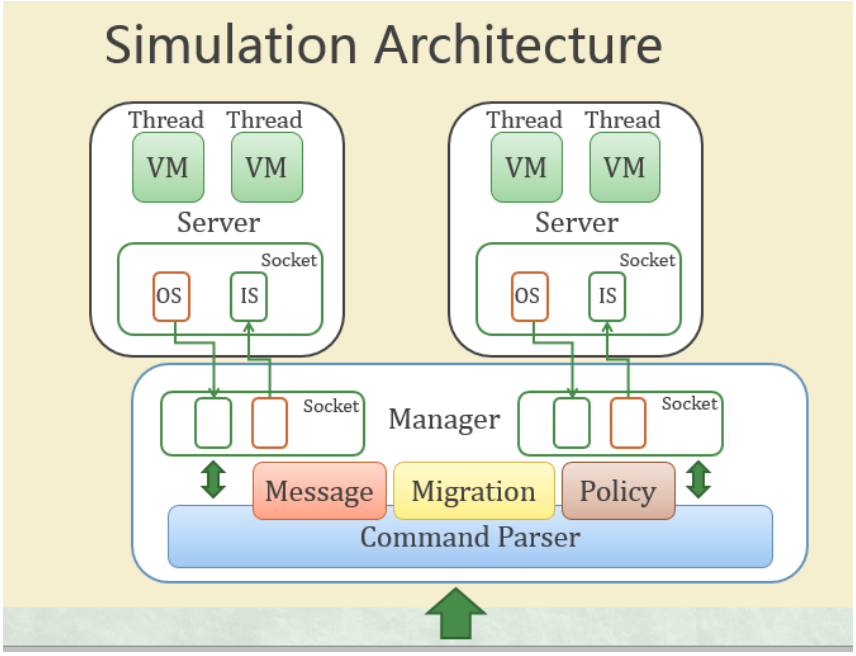
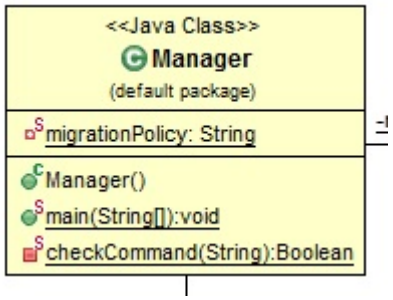


Figure 8 – Simulation of Architecture

In Figure 8, it shows the design architecture of simulation. It includes Manager, Servers, etc. In Manager, it controls the migration policy, builds messages according to user's commands and communicates with the servers, it also setup Server Socket for servers' connection. Server is the container of VMs, it is using socket to talk with Manager. Here VM is simulated by a thread; it can be stopped and resumed by Server. More detailed, check the following items.

- Migration Manager

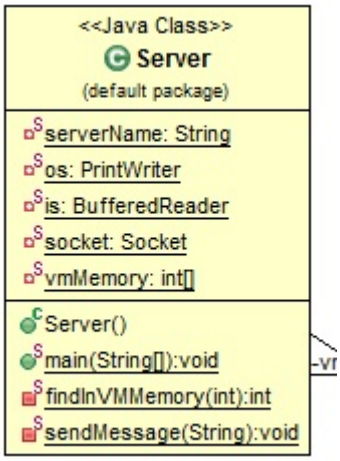


It is the controller and interface to user, which is implemented by a Java Class. Its functionalities are as following:

1. Socket Server Setup

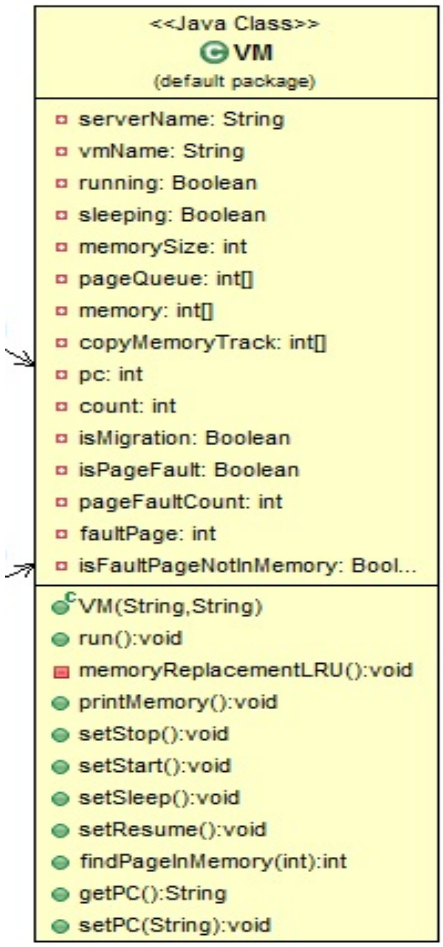
2. Command Parser
3. Set the migration policy
4. Control the Servers
5. Trigger the migration process

- Source Server and Destination Server



The server is simulated by a Java class, and it is a container to hold VMs, it has a VM Pool to store the VM information. Servers are also using socket to talk with Manager, like getting commands, and sending feedback to Manager.

- VM



VM is simulated by a Thread, it is running a LRU algorithm based on a page queue, which is integer array initialized by random numbers. VM also support interfaces to Server, so Server can start, stop, resume VMs.

- Memory

  private int memorySize = 20;

  private int[] memory = **new int**[memorySize];

We use an array to simulate memory for each VM, its size is 20 and initial default values are Zero.

- LRU replacement algorithm

1. Page Queue as the following, it is used to simulate the incoming pages.

```
private int[] pageQueue = new int[100];

Random random = new Random();

for(int i=0; i<100; i++) {
    pageQueue[i] = Math.abs(random.nextInt()) % 50;
}
```

2. LRU algorithm

```
private void memoryReplacementLRU() {
    int pos = findPageInMemory(pageQueue[pc]);

    if(pos == -1) {
        if(isMigration) {
            isPageFault = true;
            pageFaultCount++;
            faultPage = pageQueue[pc];
            try {
                System.out.println("Page fault: Page" + faultPage + " is requried.");

                while(isWait) {
                    System.out.println("Page fault: VMThread(" +
Thread.currentThread().getId() + ") is waiting!");
                    wait();
                }

                if(isFaultPageNotInMemory) {
                    if(count < memorySize) {
                        memory[count] = pageQueue[pc];
                        count++;
                    } else {
                        for(int i=0; i<memorySize - 1; i++) {
                            memory[i] = memory[i+1];
                        }

                        memory[memorySize - 1] = pageQueue[pc];
                    }

                    isFaultPageNotInMemory = false;

                    Thread.sleep(11000);
```

```
                        System.out.println("PF Handle: Page" + faultPage + " is loaded
from disk!");
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else {
                if(count < memorySize) {
                    memory[count] = pageQueue[pc];
                    count++;
                } else {
                    for(int i=0; i<memorySize - 1; i++) {
                        memory[i] = memory[i+1];
                    }

                    memory[memorySize - 1] = pageQueue[pc];
                }
            }
        } else {
            int temp = memory[pos];
            int index;

            if(count < memorySize)
            {
                index = count;
            } else {
                index = memorySize;
            }

            for(int i=pos; i<index-1; i++) {
                memory[i] = memory[i+1];
            }

            memory[index-1] = temp;
        }

        pc++;

        if(pc >= 100) {
            pc = pc % 100;
        }
    }
```

- Remote page fault handling

```
if(faultPageIndex == -1) {
    vm.setIsFaultPageNotInMemory(true);
    System.out.println("PF Handle: Page" + vm.getFaultPage() + " is not in source memory!");
        } else {
    System.out.println("PF Handle: Page" + vm.getFaultPage() + " is in source memeory!");
    CopyThread cpThread2 = new CopyThread(vm, vmMemory, faultPageIndex, true);
    cpThread2.start();}
```

7. Data analysis and discussion

I. Post-copy migration

To test if our program is able to simulate VM migration successfully, two hosts were built using our Migration Manager and a VM was created on one of them (source host). Post-copy was selected to be the current algorithm. Figure 1 shows the state of host node and destination node before the migration. After a migration request was made, the system stops and copies immediately. The VM was then resumed on the destination host. The system was able to request the page fault from the source host. The background copy also kept going until all the memory content is migrated. Figure 2 shows the states on source host and destination host after migration. It is indicated that the migration was successful and the service can still be provided during the migration, indicating that our simulation has successfully mimicked the VM migration using post-copy algorithm.

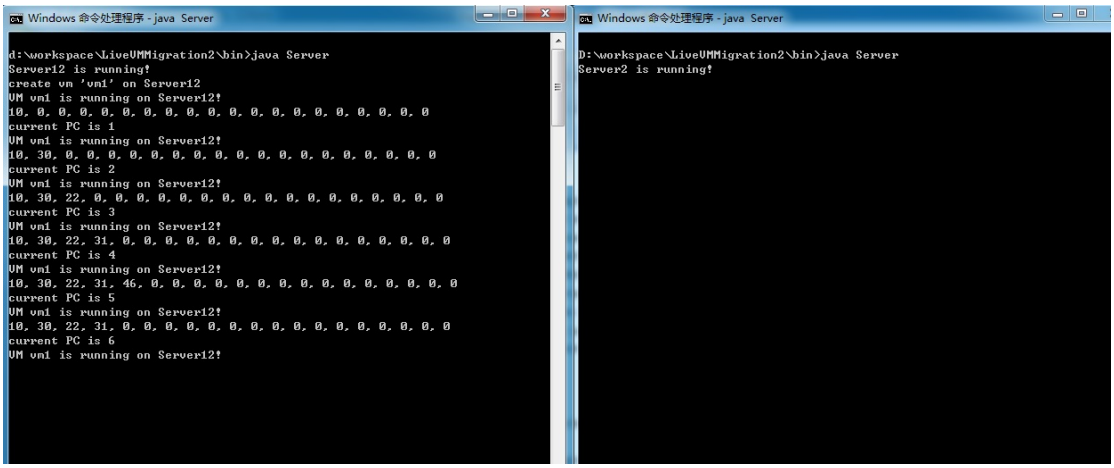


Figure 1. (a) Source host before migration. (b) Destination host beforemigration.

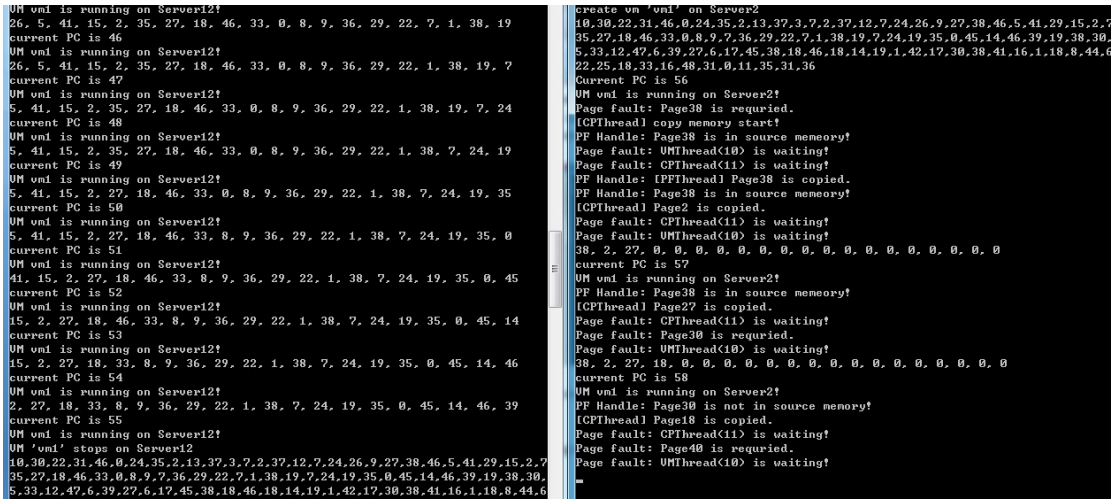


Figure 2. (a) Source host after post-copy migration. (b) Destination host after post-copy migration.

II. Lazy-copy migration

Similar steps were used to set up the host and VM for migration using lazy-copy method. When it was time to select migration method, lazy-copy was selected instead of post-copy. Before migration, the source host and destination host behave similarly as those in post-copy migration shown in Figure 1. When the migration was started, after the stop and copy phase at the very beginning of the migration similar to post-copy migration, the service was resumed on both the source host and destination host and the service on source host was slightly earlier than that on the destination host. During this time, page fault produced on the source host, extra page faults produced on the destination host, and the background copy were implemented as three different threads and started at the same time. The migration terminated after the whole memory content was migrated, and the service was then only provided on the destination host. It has been proven that our lazy-copy migration simulation was successful.

III. Total migration time comparison & Page fault comparison

Due to time limited, total migration time comparison and page fault comparison are left for our future study.

8. Conclusions and recommendations

Live migration of VM has been intensively studied over the past several years aiming for a fast and silent algorithm. The previously well-established algorithms, pre-copy algorithm and post-copy algorithm, both have limitations under certain conditions, affecting their performance. We have proposed a lazy-copy algorithm here, which theoretically reduces the amount of page faults and thus increases performance. Based on the comparison between lazy-copy and post-copy from our simulation model, it is found that lazy-copy achieved a decrease in the total migration time and a decrease in the total number of page faults, leading to a faster and more silent algorithm for VM live migration.

Further study is still needed to characterize the VM live migration using lazy-copy algorithm. Model using real computer operating systems could be used to simulate the VM. Service could be simulated using real server models. The migration could also be evaluated in more detailed aspects, including the time in different phases, the percentage of transmission of different thread, the consumed network bandwidth, etc. We have achieved a big progress in studying the algorithms for VM live migration. However, there could be even better algorithms for this purpose. To make the migration faster and more silent, it is possible to estimate the future needed pages based on the currently running programs and do background copy more selectively to further eliminate page fault. It is also possible to execute the page write only on the destination host and thus exempt those pages from migration. All the possibilities could be simulated and evaluated in our future study and gain an even better algorithm for live VM migration to facilitate nowadays cloud operation.

9. Bibliography

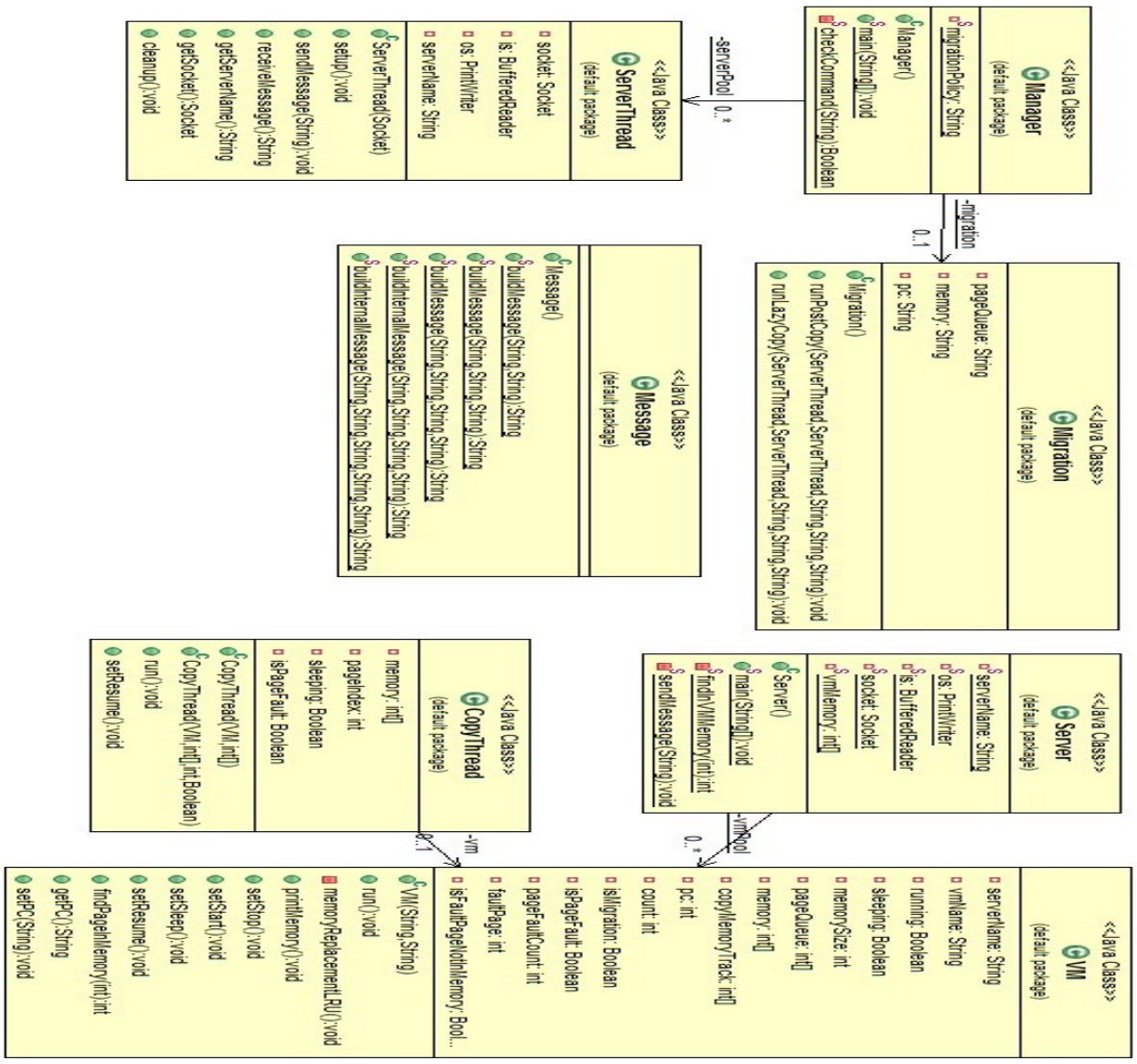
1. P. Barham, B. Dragovic, K. Fraser, S. Hand,T. Harris, A. Ho, R. Neugebauer, I. Pratt, andA. War_eld.Xen and the art of virtualization.InSOSP, 2003.
2. C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul,C. Limpach, I. Pratt, and A. War_eld. Live migrationof virtual machines.In NSDI, 2005.

3. A. Gulati, A. Holler, M. Ji, G. Shanmuganathan,C. Waldspurger, and X. Zhu. VMware distributedresource management: design, implementation, andlessons learned. In VMware Technical Journal, 2012.
4. A. Kivity, Y. Kama, D. Laor, U. Lublin, andA.Liguori.kvm: the linux virtual machine monitor. InProceedings of the Linux Symposium, 2007.
5. P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal,Z. Wang, S. Singhal, and A. Merchant. Automatedcontrol of multiple virtualized resources.InEuroSys, 2009.
6. M. R. Hines and K. Gopalan.Post-copy livevirtual machine migration using adaptive pre-pagingand dynamic self-ballooning.In VEE, 2009.
7. 7. T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi.Enabling instantaneous relocation of virtual machineswith a lightweight VMM extension.In CCGRID, 2010.
8. Wenjin Hu, Andrew Hicks, Long Zhang, Eli M. Dow, Vinay Soni, Hao Jiang, Ronny Bull, Jeanna N. Matthews. A Quantitative Study of Virtual Machine Live Migration. In CAC, 2013.
9. VIOLETA MEDINA and JUAN MANUEL GARC´IA. A Survey of Migration Mechanisms of Virtual Machines. In CSUR, 2014.
10. Changyeon Jo, Erik Gustafsson, Jeongseok Son, and Bernhard Egger. Efficient Live Migration of Virtual Machines Using Shared Storage. In VEE, 2013.
11. Jihun Kim, DongjuChae, Jangwoo Kim, and Jong Kim. Guide-Copy: Fast and Silent Migration of Virtual Machinefor Datacenters. In SC, 2013.
12. Peng Lu, Antonio Barbalace, and BinoyRavindran. HSG-LM: Hybrid-Copy Speculative Guest OS Live Migration without Hypervisor. In SYSTOR, 2013.
13. Umesh Deshpande, Xiaoshuang Wang, and KartikGopalan. Live Gang Migration of Virtual Machines. In HPDC, 2011.
14. Kejiang Ye, Xiaohong Jiang, Ran Ma, Fengxi Yan. VC-Migration: Live Migration of Virtual Clusters in the Cloud. In Grid Computing, 2012.
15. Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, Chen Yu. Live Migration of Virtual Machine Based on Full System Trace and Replay. In HPDC, 2009.
16. Robert Bradford, EvangelosKotsovinos, AnjaFeldmann, HaraldSchi¨oberg. LiveWide-Area Migration of Virtual MachinesIncluding Local Persistent State. In VEE, 2007.

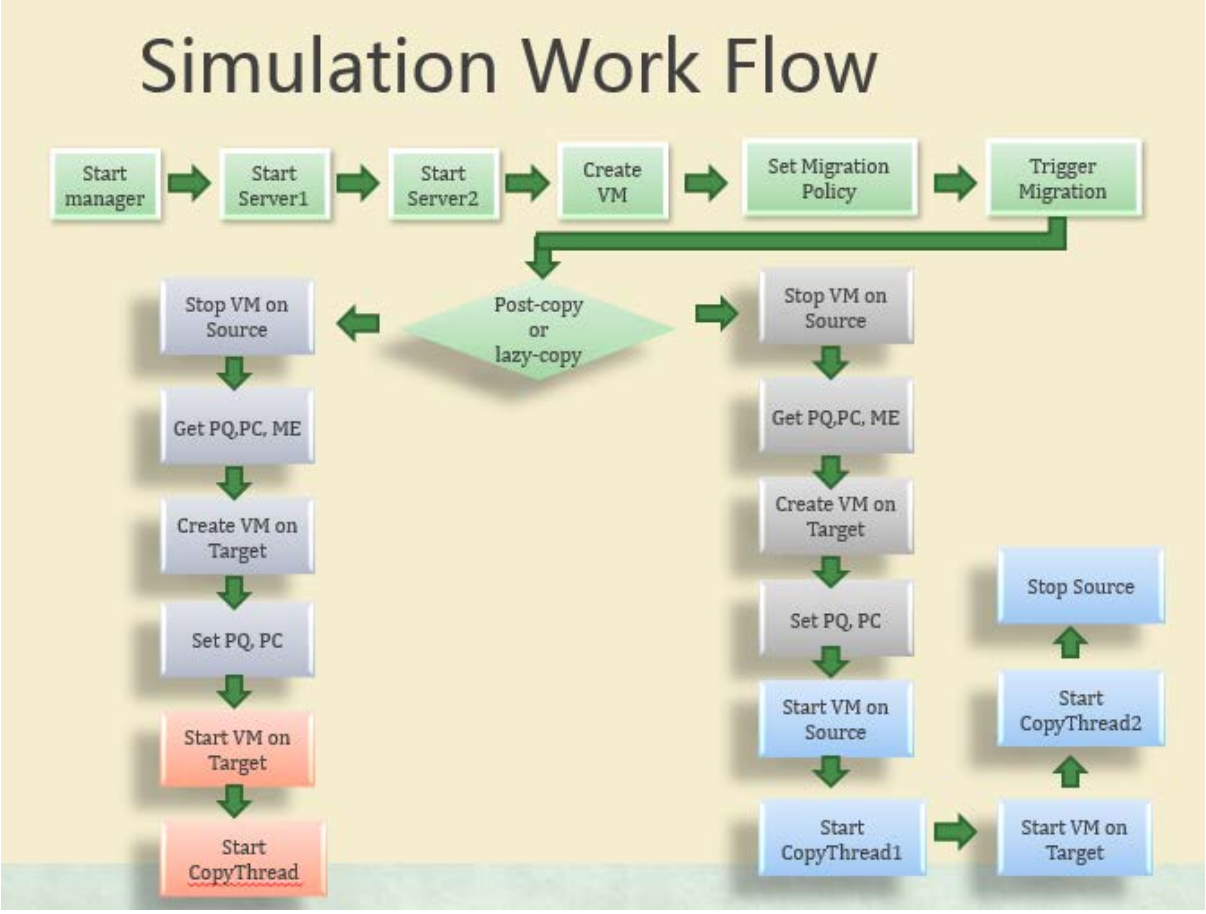
17. Ruijin Zhou, Fang Liu, Chao Li, and Tao Li. Optimizing Virtual Machine Live Storage Migration in Heterogeneous Storage Environment. In VEE, 2013.
18. Xiang Song, Jicheng Shi, Ran Liu, Jian Yang, and Haibo Chen. Parallelizing Live Migration of Virtual Machines. In VEE, 2013.
19. Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, Xiaofei Liao. Performance and Energy Modeling for LiveMigration of Virtual Machines. In HPDC, 2011.
20. Senthil Nathan, Purushottam Kulkarni and UmeshBellur. Resource Availability Based Performance Benchmarkingof Virtual Machine Migrations. In ICPE, 2013.

10. Appendices

10.1 Class Diagram:



10.2 simulation Work flow



1.P. Barham, B. Dragovic, K. Fraser, S. Hand,T. Harris, A. Ho, R. Neugebauer, I. Pratt, andA. War_eld.Xen and the art of virtualization.InSOSP, 2003.

2.C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul,C. Limpach, I. Pratt, and A. War_eld. Live migrationof virtual machines.In NSDI, 2005.

3 .A. Gulati, A. Holler, M. Ji, G. Shanmuganathan,C. Waldspurger, and X. Zhu. VMware distributedresource management: design, implementation, andlessons learned. In VMware Technical Journal, 2012.

4 .A. Kivity, Y. Kama, D. Laor, U. Lublin, andA.Liguori.kvm: the linux virtual machine monitor. InProceedings of the Linux Symposium, 2007.

5 . P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal,Z. Wang, S. Singhal, and A. Merchant. Automatedcontrol of multiple virtualized resources.InEuroSys, 2009.

6. M. R. Hines and K. Gopalan.Post-copy livevirtual machine migration using adaptive pre-pagingand dynamic self-ballooning.In VEE, 2009.

7. T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi.Enabling instantaneous relocation of virtual machineswith a lightweight VMM extension.In CCGRID, 2010.

8. Wenjin Hu, Andrew Hicks, Long Zhang, Eli M. Dow, Vinay Soni, Hao Jiang, Ronny Bull, Jeanna N. Matthews. A Quantitative Study of Virtual Machine Live Migration. In CAC, 2013.

9. VIOLETA MEDINA and JUAN MANUEL GARC´IA. A Survey of Migration Mechanisms of Virtual Machines. In CSUR, 2014.

10. Changyeon Jo, Erik Gustafsson, Jeongseok Son, and Bernhard Egger. Efficient Live Migration of Virtual Machines Using Shared Storage. In VEE, 2013.

11. Jihun Kim, DongjuChae, Jangwoo Kim, and Jong Kim. Guide-Copy: Fast and Silent Migration of Virtual Machinefor Datacenters. In SC, 2013.

12. Peng Lu, Antonio Barbalace, and BinoyRavindran. HSG-LM: Hybrid-Copy Speculative Guest OS Live Migration without Hypervisor. In SYSTOR, 2013.

13. Umesh Deshpande, Xiaoshuang Wang, and KartikGopalan. Live Gang Migration of Virtual Machines. In HPDC, 2011.

14. Kejiang Ye, Xiaohong Jiang, Ran Ma, Fengxi Yan. VC-Migration: Live Migration of Virtual Clusters in the Cloud. In Grid Computing, 2012.

15. Haikun Liu, Hai Jin, Xiaofei Liao, Liting Hu, Chen Yu. Live Migration of Virtual Machine Based on Full System Trace and Replay. In HPDC, 2009.

16.Robert Bradford, EvangelosKotsovinos, AnjaFeldmann, HaraldSchi ¨ oberg. LiveWide-Area Migration of Virtual