University of Tartu

Faculty of Science and Technology

Institute of Technology

Tambet Viitkar

Hypervisor and Virtual Machine Memory Optimization Analysis

Graduation Thesis (12 ECTS)
Computer Engineering

Supervisor:

Meelis Roos

Tartu 2018

Resümee/Abstract

Hüperviisorite ja virtuaalmasinate mäluhalduse analüüs
Käesoleva lõputöö eesmärgiks on uurida ning analüüsida hüperviisorite ja virtuaalmasinate mäluhalduse funktsionaalsust ja efektiivsust väheste mäluressursside korral. Katsetuste käigus kasutatakse laialtlevinud hüperviisoreid: VMware ESXi, Microsoft Hyper-V, KVM ja Xen. Sihiks on mõõta, kui palju mälu suudavad optimiseerimisalgoritmid hüperviisorile tagastada ning kui efektiivselt mälu koondatakse. Katsetulemuste andmed kogutakse kokku spetsiaalselt süsteemide monitoorimiseks mõeldud tarkvaralahenduste Zabbix ja collectd poolt. Analüüsi põhjal saab öelda, et kõige efektiivsem on Hyper-V, millele järgnes ESXi ja KVM. Xen hüperviisoril puudus automaatne mäluhalduse võimekus ning sellest tulenevalt ei kaasatud teda katsetustes.

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine;
T120 Süsteemitehnoloogia, arvutitehnoloogia.

Märksõnad: arvutid, hüperviisorid, virtuaalmasinad, mäluhaldus, mälu optimiseerimine, virtuaalmälu, süsteemihaldus, vmware, hyperv, kvm, xen

Hypervisor and Virtual Machine Memory Optimization Analysis

The goal of this thesis is to test memory optimization and reclamation tools in the most widely used hypervisors: VMware ESXi, Microsoft Hyper-V, KVM, and Xen. The aim is to measure how much memory could be reclaimed and optimized by different memory management algorithms across hypervisors mentioned above. Dedicated monitoring tools Zabbix and collectd are going to gather the data which will be analyzed. As a result, Hyper-V seems to be the most effective, with ESXi second and KVM falling somewhat behind in the third place. Xen failed to meet specific criteria (automated memory optimization) which rendered it impractical to include in the testing process.

CERCS: P170 Computer science, numerical analysis, systems, control; T120 Systems engineering, computer technology

Keywords: computers, hypervisors, virtual machines, memory management, memory optimiztion, virtual memory, system engineering, vmware, hyperv, kvm, xen

Contents

List of Figures

List of Tables

Abbreviations

- CPU - central processing unit
- DHCP - dynamic host configuration protocol
- DNS - domain name system
- GB - gigabyte
- GHz - gigahertz
- GiB - gibibyte
- HDD - hard disk drive
- IOMMU - input output memory management unit
- IoT - internet of things
- KB - kilobyte
- KVM - kernel-based virtual machine
- MHz - megahertz
- MMU - memory management unit
- OOM - out of memory
- OS - operating system
- POSIX - portable operating system interface
- RAM - random access memory
- RDP - remote desktop protocol
- RHEL - Red Hat Enterprise Linux
- RPM - rounds per minute
- SATA - serial AT attachment
- SSD - solid state drive
- SSH - secure shell
- VM - virtual machine
- VMM - virtual machine monitor
- VNC - virtual network computing

"And if you gaze long into an abyss,
the abyss also gazes into you."

—Friedrich Nietzsche

1. Introduction

The following thesis focuses on hypervisors and virtual machines, more precisely on *memory optimization and reclamation tools*. The goal of this thesis is to deploy most popular hypervisors, populate them with virtual machines, install operating systems focused on server applications, run stress tests on memory and gather the results after which it will be analyzed and compared. The motivation for this thesis is the growing popularity of virtualization technologies and cloud computing. Memory is one of the essential components in computers, and efficient memory management is a key to lower hardware costs, and higher consolidation ratios[1]. This thesis analysis should answer following questions: how different guest operating systems and hypervisors behave in low memory conditions, how efficiently can they reclaim memory and if there are any severe flaws, to what extent and what are the possible causes.

The thesis consists of following chapters:

1. Introduction
2. Virtualization
3. Hypervisors & Virtual machines
4. Virtual memory & Optimization
5. Methodology & Laboratory
6. Analysis
7. Summary

The second chapter, *Virtualization*, gives a short overview of the history of virtualization and where it stands today. Third chapter *Hypervisors & Virtual machines* introduces type 1 and type 2 hypervisors; what is resource allocation; hypervisors available today and what exactly is a virtual machine. The fourth chapter, *Virtual memory & Optimization*, consists of a brief review of virtual memory, clarifies confusing terminology and exhibits modern memory optimization techniques. The following chapter *Methodology & Laboratory* specifies the hardware (server, hard disks, network device) and software (hypervisors, guest operating systems, monitoring systems) that will be used to assemble a testing lab and how it all connects. In the final chapter *Analysis* gathered data is examined and compared. As this is a scientific paper, it will try its best to be unbiased, but as a human is its author, then there is a risk of a possible unconscious and tilted preferences[2]. The thesis ends with a summary that concludes the findings and gives an assessment: how successful were the tests, did memory optimization work, how well did it work, did memory reclamation fail at any point, did any of the virtual machines or hypervisors crash and what research could be done in the future.

The following expectation is made in advance and verified as thesis progresses through the testing and analysis: commercial tools are easy and fast to setup and configure, leaving more time to test hypervisors and virtual machines. Similar results are anticipated across hypervisors, and guest machines — differences in results are presumed to be minuscule.

[1] Consolidation ratio is the number of VMs on a server

[2] Sincere apologies in advance.

On a side note. As technology is changing at a rapid pace, it becomes ever more difficult to find informative and up-to-date sources in traditional book form. Therefore this thesis relies heavily on online sources, not because of deliberate choice, but due to the awareness that online sources are updated and created on a more regular basis. It is up for debate how books compare to web sources — for example, quality versus quantity, but that discussion is for another day. There are however two books worth a mention as they were used exclusively throughout this thesis:

- *Virtualization Essentials* (2nd Edition) by M. Portnoy
- *Modern Operating Systems* (4th Edition) by A.S. Tanenbaum & H. Bos

1.1 Connection to computer engineering

At first sight, it might not make much sense for computer engineers to take an interest in virtualization. After all, what has it got to do with electrical engineering, hardware, robotics, computer vision, embedded systems and so forth?

It is arguable that virtualization can be to computer engineering what networking is to the Internet of Things. Many do not appreciate the delegate world of network engineering that is one of the cornerstones that makes the Internet of Things possible in the first place; it is taken for granted.

M. Tim Jones wrote an excellent article for IBM, *Virtualization for embedded systems* (2011) laying out why virtualization in embedded systems *can be* and *is beneficial*. He states that a hypervisor can considerably enhance embedded system's efficiency (real-time capabilities), add an extra layer of security, better communication, isolation, and reliability. Embedded hypervisors are used to create platforms where legacy applications can run concurrently with newer software packages. For example, PikeOS is an embedded hypervisor used *extensively* in the field of avionics for safety-critical applications and has made its way into automotive systems [15].

Embedded hypervisors are out of the scope of this thesis as it is a more advanced topic. Server virtualization is currently the trend with vast amounts of resources available both online and on paper. It is an excellent introduction to the world of virtualization on which to lay a strong theoretical and practical foundation and carve the way for more high-level and in-depth research and projects.

In addition to embedded systems, another example presents itself in the form of *cloud robotics* that could employ virtualization to create virtual environments, robotic ecosystems and off-load computationally expensive operations to massive data centers. Moving robotics to the cloud provides worldwide access to these machines from almost any location and allows to design and assemble smaller and mobile devices. Building something that is small, computationally powerful with reduced strain on onboard power supplies can stifle mobility and considerably increase design and manufacturing cost[20, 29].

A high-level configuration of hypervisors requires a deep understanding of the underlying physical hardware, e.g., one has to know how a computer works down to the metal to

make the best out of virtualization. That is something that computer engineers could excel. After all, virtualization is nothing more than another tool in the toolbox that is readily available and can be utilized to create better, more resource efficient and secure products, services and systems.

2. Virtualization

2.1 What is virtualization?

Whenever there is a discussion about computers, networks, sharing resources and utilizing those very resources to their fullest, it usually does not take long until the word *virtualization* gets mentioned.

What exactly is virtualization? Due to its popularity, virtualization has been described and defined in many different ways by various organizations and individuals. WMware, one of the leading companies specializing in cloud infrastructure and digital workspace technology, describes it as follows. *'Virtualization is the process of creating a software-based, or virtual, representation of something, such as virtual applications, servers, storage, and networks. It is the single most effective way to reduce IT expenses while boosting efficiency and agility for all size businesses.'* [33].

RedHat, the world's leading provider of open source solutions, defines it as *'Technology that helps to create useful IT services using resources that are traditionally bound to hardware. It allows using a physical machine's full capacity by distributing its capabilities among many users or environments.'* [28].

There are many definitions in similar vain available both online and print form. More broadly speaking, virtualization is an intricate part of the modern information age. Not only does commercial business-world depend on virtualization, but it is also a part of our everyday lives and services we use, love and are accustomed to. Even if we are not aware of its presence on a daily basis, virtualization has changed how computers: hardware, software, networks, and services are distributed and implemented these days.

2.2 Where did it start?

Just like many technological achievements in use today (e.g., touchscreens[1]), virtualization is not a new concept, and its roots reach back to 1960s and 1970s.

Gerald J. Popek and Rober P. Goldberg codified the framework that describes the requirements for a computer system to support virtualization. Concepts from their article that was released back in 1974 called *"Formal Requirements for a Computer System to Support Virtualization"* are still widely in use even today [25].

IBM was one of the first to utilize virtualization, specifically on their mainframe IBM System 360/67. IBM called it *platform (system) virtualization*, and it aimed to share computer's resources with multiple users (time-sharing). It ran CP/CMS operating system [14, 25, 32].

[1] https://goo.gl/eyYE1r

2.3 Types of virtualization

Virtualization classifications vary somewhat depending on the (online) resources, but the most common and well-known types are:

- Server virtualization
- Desktop virtualization
- Operating system virtualization
- Network virtualization

This thesis focuses on *server virtualization*. Servers are high-end computers designed to process vasts amount of data. Server virtualization means that a physical server is partitioned into multiple sub-machines (VMs) within the virtualization software (hypervisor). The hypervisor provides VMs with access to the host's hardware by allocating a portion of available resources to each VM — defined by the system administrator in VM configuration files.[38].

3. Hypervisors and Virtual Machines

3.1 Hypervisors

A hypervisor[1] acts as an authority — its job is to allocate resources between one or more virtual machines (instances; guests) that operate on top of it by continually monitoring guests and their resource requirements and active utilization rates in order to redistribute resources and boost system's overall efficiency. Without hypervisors, it would be difficult to take full advantage of the modern processing power and memory resources simply because most applications only use a fraction of what is available, leaving a significant percentage of resources unadapted [25].

3.1.1 Type 1

Type 1 hypervisor, also known as a *bare-metal* hypervisor, (figure 3.1) operates directly on the physical hardware. Type 1 hypervisors are much more efficient at deploying resource to VMs compared to type 2 hypervisors; mainly because of the absent OS layer that is present in type 2 that induces additional overhead. Low overhead makes for a more productive system and enables to create and run more VMs concurrently on a single host — increasing the consolidation ratio [25].

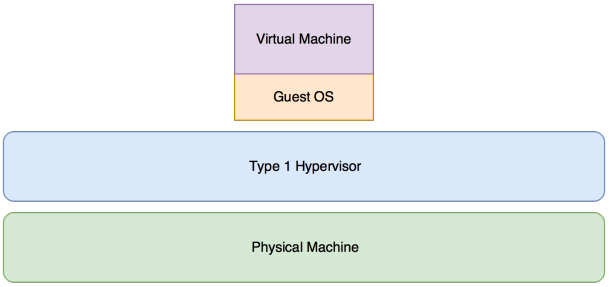

Figure 3.1: Type 1 hypervisor

Type 1 hypervisors are also deemed more secure compared to type 2. All VMs and guest operating systems within them are independent of each other, unable to have any effect on the hypervisor. A guest can only affect and damage itself without escaping the virtual machine's realm [25].

3.1.2 Type 2

Type 2 hypervisor, on the other hand, (figure 3.2) behaves more like an everyday application that runs on the already existing operating system through which hardware resources are accessed. Type 2 hypervisors are substantially easier to install and run because the operating system incorporates much of the hardware functionality (communication, resource allocation, support for wide range of hardware and so forth) that is required by the

[1] *"Initially, the problem that the engineers were trying to solve was one of resource allocation, trying to utilize areas of memory that were not accessible to programmers. The code they produced was successful and was dubbed a hypervisor because, at the time, operating systems were called supervisors and this code could supersede them."* — M. Portnoy [25]

virtualization software. The underlying OS that hosts type 2 hypervisors also consumes physical resources itself. That can drastically affect both the efficiency and the level of resources that are available for guest machines [25].

Type 2 hypervisors are also more susceptive to failures which renders them less reliable. Whatever happens to the host will affect every virtual machine[2]. Some operating systems require a system reboot after an update, even forcing it upon the user, which means that all active virtual machines need to reboot as well [25].

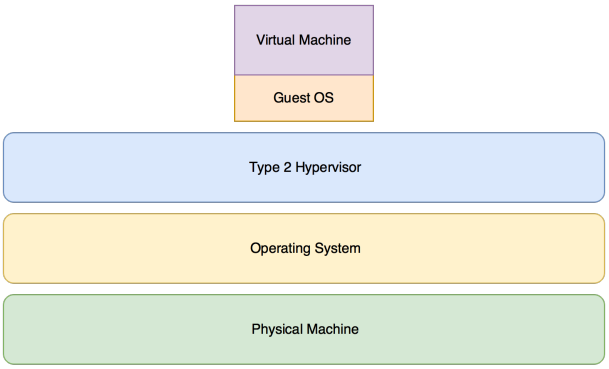

Figure 3.2: Type 2 hypervisor

3.1.3 Resource Allocation

Hypervisor functions as a workload balancer. Applications on guest systems make countless requests to the hardware. Since VMs do not have any access to host's components[3] then a hypervisor must step in and manage all those requests promptly and provide sufficient supplies. To do that resource scheduling processes are used. Depending on the hypervisor vendor it might even be able to prioritize mission-critical virtual computers over low priority systems and dynamically reallocate resources on the fly without any input from system administrators [25].

A certain amount of available physical memory will be reserved and used by the hypervisor. Depending on memory availability and the hypervisor this can be up to 20% of total RAM, but that is a rare case. Modern hypervisors require far less than that: VMware ESXi and Windows Server 2012 R2 with Hyper-V require roughly 2 GB of RAM to operate. In addition to hypervisors, virtual machines reserve a certain amount of memory too to power up and keep themselves running. This sort of memory reservation is called *memory overhead*, and although it is a crucial concept to understand and take into account in production environments, it is not relevant in the scope of this thesis [8, 25, 34].

When configuring virtual machines, it is a good idea to have some *headroom*, i.e., not appropriating all the resources. If a number of VMs work on a task that requires a considerable amount of memory those resources should be promptly available. Lack of memory can lead to all sort of problems that can cause slowness, system errors and crashes: usually in the virtual machine, but in worst case scenarios in the hypervisor itself which could affect every virtualized appliance [30].

[2]This is not to say that type 1 hypervisors do not need updates or never fail. Instead, their design should prevent hiccups of this kind as much as possible.

[3]In most cases VMs are not even aware that they are virtual.

3.1.4 Options for Type 1 Hypervisor

VMware ESXi

VMware was the first company that made x86 architecture virtualization available commercially (1998), holding close to 70% of the global market today. ESXi architecture's core is its VMkernel (figure 3.3) that includes all the tools to create and manage virtual machines and allocate resources. It additionally supports third-party modules (i.e., hardware drivers). The kernel is a POSIX-like operating system that performs analog to other operating systems, but its primary goal is to host virtual machines. It includes tools to handle resource scheduling, I/O stacks, device drivers and so forth. Out of all the hypervisor developers, VMware could at this time still be considered the most mature [4, 25].

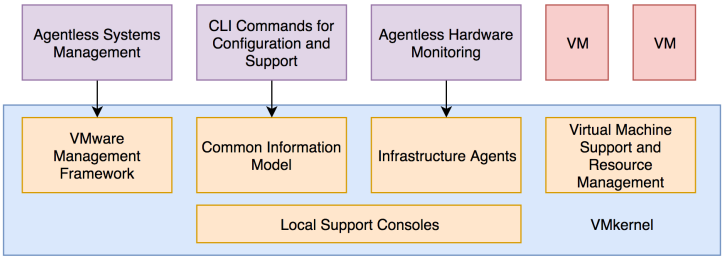

Figure 3.3: VMware ESXi architecture

Hyper-V

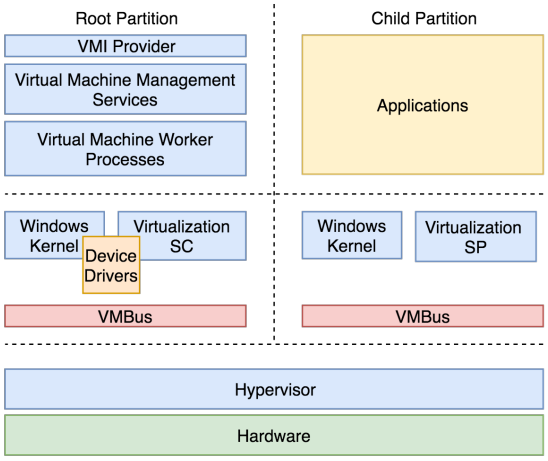

Figure 3.4: Hyper-V architecture

Microsoft has been developing its type 1 hypervisor (figure 3.4) since 2005 and released it in 2008. Today they control about 20% of the market. Compared to VMware ESXi, Hyper-V requires a *parent partition* (root partition) that has access to the hardware and holds the virtualization management stack. A partition alone is not enough, and Windows operating system is required[4]. Hyper-V is part of Windows as an optional feature, and no extra software is required. Hyper-V manager handles virtual machine (*child partitions*) creation and management. Child partitions have no direct access to the physical host[5], and all the communication and resource allocation goes through the

[4]Windows Enterprise, Professional, Education or Server [6]
[5]Hyper-V supports isolation as a partition

Xen

Keir Fraser and Ian Pratt are the creators of Xen hypervisor which was part of a research project started back in the late 1990s at Cambridge University. Xen is available both as an open-source project and a commercial edition known as Citrix XenServer. One of the primary components of Xen's architecture (figure 3.5) is Domain 0's (Dom0) guest. As the host machine running Xen hypervisor boots, Dom0 starts along with it. Domain 0 is the intermediate device between host's hardware resources and VMs. If a guest OS makes a request, it goes through the Xen hypervisor straight to the Domain 0 which grants access to resources. As Domain 0 is a guest system too, it requires additional configuration (best practices) to achieve a more stable and prolific system [25, 45, 46].

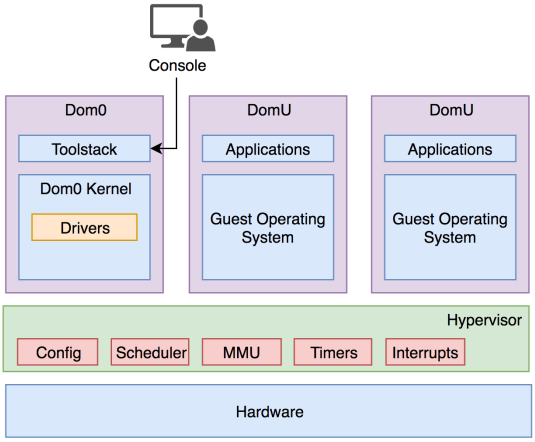

Figure 3.5: Xen architecture

Kernel-based Virtual Machine

Avi Kivity, an Israeli software developer, is the original author of KVM but is no longer actively involved with its development. Announced in 2006, KVM became part of Linux's kernel[6] in 2007. As KVM is integrated to Linux (figure 3.6), it has access to the memory manager, process scheduler, network stack and other parts of the operating system. Virtual machines created within KVM are regular Linux processes with a dedicated virtual hardware which can be managed through QEMU tools or with the libvirt-based stack. KVM is actively gaining popularity and in the future might become one of the leading contenders to VMware. Being an open-source hypervisor creates a state where the hypervisor becomes a commodity, and the management tools, their features and ease of use become the selling point [17, 23, 26, 27].

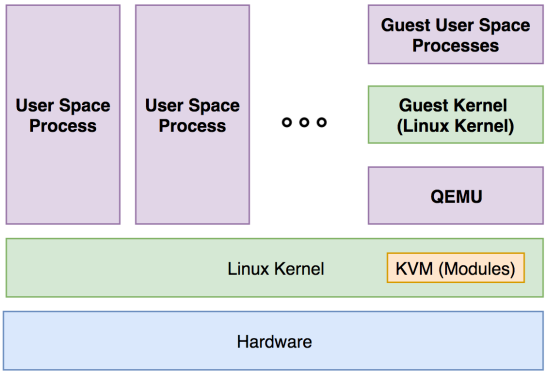

Figure 3.6: KVM architecture

[6]Linux 2.6.20 or newer.

3.2 Virtual Machines

A virtual machine behaves more or less like a conventional physical computer. The difference being, and as the word *virtual* might insinuate, it is an implicit representation of something that exists in the real world. Unlike a physical device which can only run one operating system at a time, it is possible to run multiple virtualized systems on a single host, limited only by its physical resources and capabilities. Virtual machines can run different operating systems, set of applications and serve unique purposes [25].

A virtual machine comprises a set of files: customizable configuration file and virtual disk file(s). The configuration file defines the hardware used by the VM: the central processing unit, number of cores, amount of memory, network interfaces and so forth. The virtual disk file, on the other hand, contains the guest operating system, all the required applications, and other system and user files. As configuration files remain the same, it enables to move machines across different hardware and virtualization platforms — making VMs very mobile [25].

4. Virtual memory and optimization

"More than any other resource, memory is the one with the largest impact on how well or how poorly the virtual environment will perform."

—Mike Portnoy, *Virtualization Essentials*

Memory, also known as *random access memory* is the essential part of any computer system. It is the container that stores data that is accessed most often by the operating system and active applications [25]. It is imperative to recognize that memory is *not* storage and to this day there is plenty of confusion between the two. This thesis focuses *exclusively* on (random access) memory.

Compared to hard disks and solid state drives, memory is many times faster and presents a more flexible model compared to accessing data from storage devices. Speed comes with a price tag, literally. RAM is expensive, and it can become scarce. Therefore, efficient memory optimization and strategic resource allocation are vital to run multiple VMs on a single physical machine [25].

In VMs, memory is made available by the hypervisor (abstracts memory) who functions as an intermediate player between the physical hardware and virtual machine(s). A host might have a total amount of 16 GB of RAM (figure 4.1), but if the VM has been configured to use 2 GB, then it is only aware of those 2 GB[1]. Remaining RAM can be applied to other guests or as a possible memory expansion to the already existing VMs if demand for it should arise.

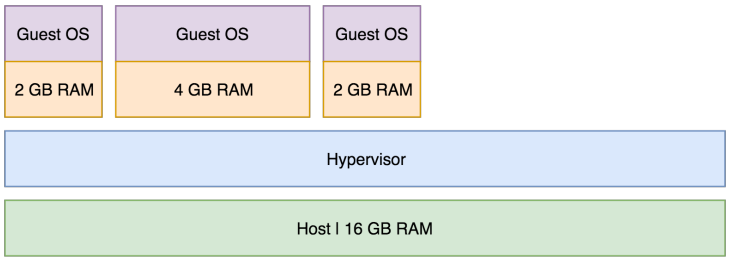

Figure 4.1: Memory allocation

Allocating too much memory can create paucity in guests and lead to excessive paging which in turn affects the whole virtualization domain. Being too moderate while allocating RAM can cause resource wastage — the exact problem hypervisors were meant to solve.

4.1 A brief overview of virtual memory

In this section, it is reviewed how modern operating systems handle the concept of *virtual memory*. It must be made clear that virtual memory is *not* a feature of hypervisors nor virtual machines; it is part of operating systems. The idea being that every programme (process) has its virtual address space (figure 4.2) that comprises the physical memory and secondary storage. That address space gets split into fixed-size pieces (most commonly 4 KB, but not strictly bounded to) called *pages*. Its counterpart in RAM is a *page frame*.

[1]Not entirely accurate in Hyper-V's case and it will be addressed later in the thesis.

The mapping (address translation) from virtual address space to the physical address space is handled by MMU and information how to map virtual pages onto page frames is stored in a *page table* [31].

An example: Application runs, storing the most important pages in the physical memory. If the application requests data from a page that is mapped to physical memory, then the data transfer happens almost instantly. However, if the referenced part of its address space is not in physical memory, a page fault is thrown. The software is put on halt, operating system analyses the memory regions assigned to this particular application and finds the least used frame. That frame gets transferred to a secondary storage unit, and the missing piece of data required by the application is loaded from disk and stored in memory. The page table is updated, software notified, and the process continues where it left off [31].

In summary, virtual memory consists of physical memory which extends its reach to secondary storage if random access memory is in short supply. The amount of secondary storage assigned to aid memory is set by configuring a *page file*.

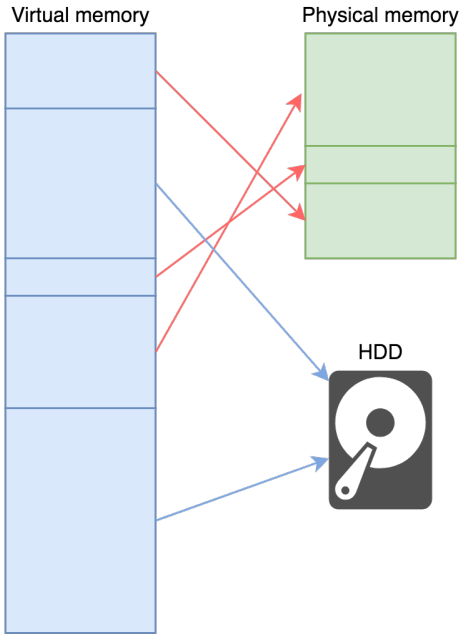

Figure 4.2: Virtual memory

4.2 Swapping vs. Paging

Swapping and *paging* are used as interchangeable terms as they describe moving processes from primary memory to a secondary storage unit. However, there is a clear difference between these two terms in how they operate.

Swapping is a method of moving a particular process, its entire data set to a swap file or (swap) partition. For swapping to be effective, the process should be idle; otherwise, there is a high probability that the process is requested again and has to be loaded back to memory rapidly, rendering swapping an unavailing operation. Excessive swapping can have a devastating effect on the system's overall performance [2, 44].

Paging is more relatable to virtual memory as it is quite literally moving pages from system memory into a page file. In paging, only a portion of an active process; less used data, is relocated. Paging also allows OSs to target specific sections of running applications that have been idle for some time and move them out of memory, making room for other data [2, 44].

Modern operating systems use paging as their default method[2], and throughout this thesis this term is going to be used [16].

[2]Linux has traditionally referred to swapping as paging.

4.3 Memory Optimization

Memory Optimization				
Hypervisor	Overcommiting	Ballooning	Page Sharing	Compression
ESXi	✓	✓	✓	✓
Hyper-V		✓	✓	
KVM	✓	✓	✓	
XEN		✓	✓	

Table 4.1: Memory features of hypervisors

4.3.1 Overcommiting

Overcommitting is a method to allocate more memory to virtual machines that is available on the host. In other words, the hypervisor is memory-overcommitted if the sum of assigned memory across all active virtual machines exceeds the possible physical memory. Although memory overcommitment is a very abstract idea, it provides an additional method to appropriate and redistribute resources, enhancing efficiency and increasing the consolidation ratio [1, 25].

It might seem like an odd move to allocate memory that does not exist. However, idle VMs might not utilize all the memory accessible to them. Overcommitment allows to extricate those resources and reallocate them to other VMs accordingly - creating a dynamic buffer. That makes memory reclamation techniques an essential part of overcommitment but as it has more to do with consolidation ratio it will not be investigated further in this thesis [1].

4.3.2 Ballooning

Ballooning is a memory reclamation technique that allows the hypervisor to retrieve a certain amount of memory from VMs if host's physical memory usage has reached a certain threshold (i.e., low memory state). It is important to heed that every VM is an isolated instance and guests are not aware of their *virtualized-ness*. It is therefore impossible for guests to detect when the host is low on memory and the whole domain is in danger of crashing. To solve this problem a *deus ex machina* in the form of memory ballooning was introduced.

Ballooning is made possible by balloon drivers which are usually part of the hypervisor's guest drivers. In a low memory state, the host notifies the guest's balloon driver, sets a target size and waits. Target balloon is the amount of memory the host strives to retrieve from a VM. Once ballooning is triggered, it inflates (figure 4.3) and forces the OS to reorganize its memory content. Guest OS must utilize its optimization algorithms to figure out how to best accommodate to the inflating balloon: remove unnecessary information from random access memory, store less crucial data in a page file and so forth.

Memory retrieved by ballooning can be reallocated to more resource intensive or higher priority guests. In ideal cases ballooned memory is returned to their rightful owners once resource intensive operations have completed. As a side note: extensive ballooning can lead to widespread paging which can drastically decrease performance [25, 36].

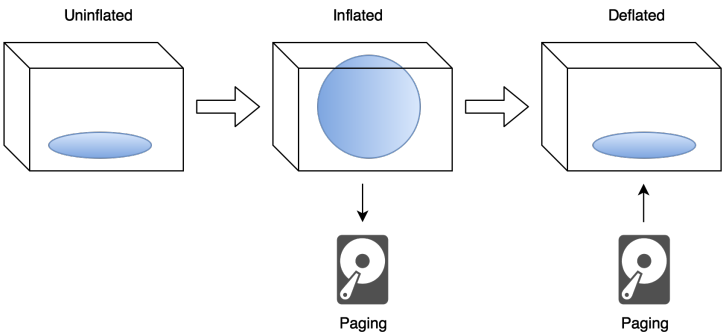

Figure 4.3: Memory ballooning

4.3.3 Page sharing

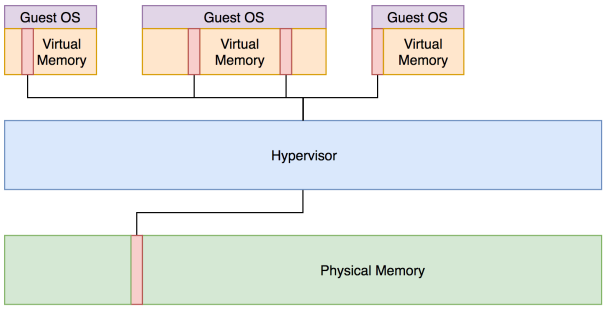

Figure 4.4: Page sharing

Plenty of data stored in RAM is never altered by the OS nor by the user. As guests populate the memory with system and application data, many of those pages are homogeneous across multiple VMs — exhausting memory by containing duplicates of the same data. Page sharing aims to remove duplicates (figure 4.4) by continually analyzing pages and keeping one copy that is shared among numerous VMs, consequently saving RAM [25].

In a situation where a VM requests write permission to a page that has a shared tag associated with it, the hypervisor will create a copy of that page and assign exclusive access rights to the VM that requested it in the first place. This process is called *copy-on-write* [25].

4.3.4 Compression

Compression is a memory optimization tool that aims to decrease the level of paging and enhance performance. A compression cache gets created by the hypervisor. If paging is about to occur those pages are analyzed and if possible compressed and stored in that memory cache for future use. Idealistically this should somewhat decrease the negative aspects of paging as it is computationally costly and it would be far easier to fetch and decompress data from the compression cache instead than loading it from an extensively slower secondary storage device [25, 35].

5. Methodology and laboratory

5.1 Methodology

The goal of this thesis is to *test memory optimization and reclamation tools* in VMware ESXi, Microsoft Hyper-V, KVM, and Xen — measure how much memory can hypervisors reclaim from VMs; how much RAM are VMs willing to return to the host; how much can optimization algorithms decrease memory usage in VMs. To do that each hypervisor is going to be populated by four guest machines: Ubuntu 16.04, CentOS 7, Windows Server 2012 R2 and Windows Server 2016. Each operating system is to be given a unique hostname and prepared to run a memory stress tool and data gathering software.

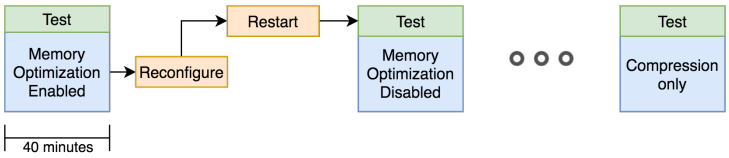

Figure 5.1: Test procedure

Each hypervisor is to be tested with, and without any memory optimization tools enabled and one memory optimization tool at a time. From now on, these test scenarios will be referred to as *conditions*.

Each test condition must follow the guideline:

- A single condition is to be tested three times in a row (iterations)
- Stress tools must start simultaneously (a small margin of error allowed)
- Each iteration should run for ten minutes
- After each iteration there should be a five-minute cooldown to let the system regain stability
- Roughly 95% of memory should be allocated by stress tools to provoke memory optimization and reclamation tools and minimize undesired paging
- Test results should be gathered automatically by dedicated software solutions
- Host must be rebooted before the next test condition

An example: VMware ESXi supports three memory optimization/reclamation tools: memory ballooning, page sharing, and memory compression. Therefore there are five discrete conditions: memory optimization enabled, memory optimization disabled, ballooning only, page sharing only, memory compression only. Each test condition runs roughly for 40 minutes — three iteration (10 minutes each) with five-minute cooldowns between each iteration excluding the last one.

5.2 Laboratory

5.2.1 Hardware

Server

Sun Fire X2250 manufactured by Sun Microsystems (provided by supervisor Meelis Roos) is the server unit on which memory optimization testing took place. It holds two Intel Xeon E5405 64-bit quad-core CPUs with a base frequency of 2.00 GHz and bus speed of 1333 MHz. It has a 12 MB (cache per processor 2 x 6 = 12 MB) Level 2 cache memory. It supports up to 32 GB of RAM (8 slots); fully utilized in this particular unit. Intel Xeon E5405 *does not* support following technologies: Intel Turbo Boost Technology, Hyper-Threading Technology, VT-x with Extended Page Tables [5, 13].

Hard Disk Drives				
ID	Manufacturer	Size (GB)	RPM	Hypervisor
EM74	Western Digital	500	7200	ESXi 6.5
EFDN	Western Digital	500	7200	Hyper-V 2012
8Y4J	Western Digital	500	7200	Xen 4.8
H4T7	Toshiba	500	7200	KVM

Table 5.1: Hard disk drives

Storage

Hypervisors were installed on one of the three Western Digital manufactured Enterprise Class Certified by Dell and one Toshiba/Hitachi HDDs (provided by Meelis Roos). Each SATA drive is capable of storing around 500 GB of data with platter rotation speeds up to 7200 RPM. Drives were assigned unique identification codes (table 4.1) to help distinguish them from one another. Multiple disks allowed to create strictly segregated test environments for each hypervisor and switch between them in a short amount of time. An additional benefit was the ability to return to these environments and repeat tests if necessary. With a single hard drive, test environments would have been impossible to set up and run in a reasonable amount of time.

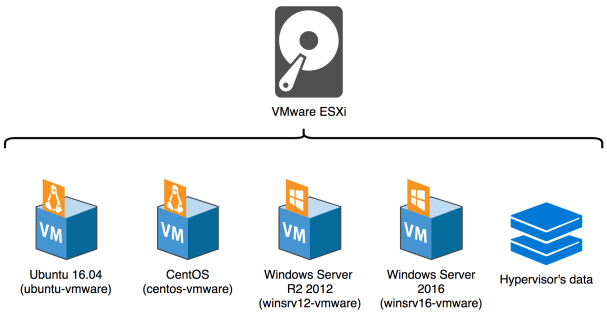

Figure 5.2: ESXi disk layout

Each disk has a similar layout (figure 5.2): four operating systems with unique hostnames to differentiate between them in the configuration, testing, data gathering, and analysis phase. Linux systems were assigned around 20 GB of hard disk space, and for Windows Server variants 40 GB of space was allotted. Rest of the space was either used by the hypervisor (configuration files, data files and so forth), page file or left untouched.

Network

A dedicated local area network was built (figure 6.10) to connect all the physical hardware, hypervisors, virtual machines, monitoring and data gathering services. Linksys WRT54GS[1] served as the pivotal piece assuming the role of a router, switch, and DHCP server. DNSMasq (built-in to WRT54GS) provided DNS services. This network drastically simplified both the setup and management of all the devices and services, especially data gathering, that were integral throughout the test phase.

5.2.2 Software

Hypervisors

VMware ESXi 6.5 installation and configuration was simple. Being the first in the commercial hypervisor market has its benefits — ESXi works out-of-the-box. It comes with a web-based administration panel which frees it from platform dependent restrictions and allows the administrator to configure the system from whatever platform he or she chooses as long as it supports a modern web-browser.

Due to server's hardware limitations, namely lack of IOMMU, it was impossible to install the latest Microsoft Windows Server 2016 with Hyper-V. Hyper-V was very persistent that this requirement is a *must-have* with no exceptions. Windows Server 2012 R2 with Hyper-V[2] was set up instead. Overall very simple to set up — user-friendly, but less configurable, at least from the graphical user interface, compared, for example, to VMware ESXi 6.5.

Installing Kernel-based Virtual Machine was also simple as it is part of the modern Linux kernel. Unfortunately, out-of-the-box it does not include sophisticated tools to manage memory optimization automatically, for example, memory ballooning [21]. Fortunately, there is a wide variety of management tools available for KVM, both open-source and commercial [22]. Based on Google's search engine results, oVirt Management Platform (built on top of KVM) was chosen as it supports automatic ballooning and KSM; has excellent documentation and in addition to KVM models on the RHEV-M[3] [24].

Xen was the most challenging hypervisor which in the end failed to meet specific requirements and was not tested. Similarly to KVM, Xen supports ballooning, but in its genuine form, that process is manual. Of course, there are more capable frontend solutions available that include automation, but in Xen's case, it is troublesome to find good open-source alternatives. Xen is mainly known for Citrix XenServer, and that is a license based hypervisor. It is possible to request a trial license for the latest XenServer, but as the current testing equipment is somewhat older, this hypervisor's installation failed. Older versions of XenServer are available, but it is not possible to request a trial license.

[1] Firmware version DD-WRT v24-sp2 (10/10/09) std.
[2] Hyper-V is also available as a standalone command line based version.
[3] Red Hat Enterprise Virtualization Management.

Operating systems

Following operating systems were installed in each hypervisor:

- Ubuntu Server 16.04
- CentOS 7
- Windows Server 2012 R2
- Windows Server 2016

At the time of composing the current thesis, those OSs were the latest and most commonly referred to and applied in production environments around the globe. It presented an excellent opportunity to compare two open-source OSs against proprietary solutions and furthermore compare systems in those two categories against each other.

Ubuntu is based on the Debian architecture and developed and published by Canonical Ltd. It is a ubiquitous choice for both desktop applications and servers alike as it supports prominent processor vendors, including Intel, AMD, and ARM. Ubuntu 16.04 LTS (released on April 21, 2016), codenamed Xenial Xerus has an expected end of life date of April 2021 [37, 39].

CentOS distribution stems from Red Hat Enterprise Linux. Compared to RHEL, it is available at no extra cost and strives to provide enterprise-class functionality just like its origin but its entirely community supported. Current main versions are 6.9 and 7. Version 7 (released on July 7, 2014) maintenance updates will be available until June 2024 [3, 43].

Windows Server 2012 is the sixth release in the Windows Server branch released on September 4, 2012. Mainstream support is scheduled to end in October 2018. Windows Server 2016 was developed alongside with Windows 10 and released on October 12, 2016. Mainstream support is available until January 2022 [41, 42].

Stress tools

Stress-ng[4] runs on GNU/Linux. It is an updated version of the popular *stress* tool which is used to accentuate pressure on a computer in various ways (e.g., CPU, GPU, memory, I/O) and it has a dedicated toolset for *virtual memory*. Linux's kernel has a feature that attempts to kill any process that completely hogs the system (i.e., out of memory). Stress-ng's countermeasure to this is to run stress workers as separate processes and if they happen to get terminated the main program restarts them [11, 18, 19].

Following are the most relevant stress-ng parameters in the scope of the current thesis:

- *–vm* - number of workers (default: 1)
- *–vm-byte* - bytes allocated per worker (95% of total VM's RAM)

[4]Version 0.08.16.

- *–vm-method* - stress methods (default: all)
- *-q* - quiet mode
- *-t* - test duration (seconds)

A console-based program named TestLimit by Sysinternals (Mark Russinovich) was used in Windows Server. There is not much information available about TestLimit, other than it is a rather popular and recommended choice to stress and leak memory in Windows. Following parameters were used:

- *–d* - leak and touch memory in a specificied MBs
- *–c* - count of number of objects to allocate

A small script[5] was written in Python 3 programming language to aid in the memory optimization test phase. The script requires four parameters; an explanation will follow after this paragraph. The Python script is platform-aware, and depending on the operating system it will start one of the stress tools with the parameters discussed above and manages cooldowns between iterations. As simple as this script is, it was an indispensable tool without which testing would have been very cumbersome.

Following are the Python's stress test management script's parameters:

- *i* - number of test iterations (default: 3)
- *t* - duration of a single interation (default: 30 s)
- *r* - duration of cooldown (default: 10 s)
- *mem*[6] - memory utilization in percentage (default: 0.5)

```
witkaer@witkaer-home:~/Nextcloud/vm-memory-management-thesis/Scripts$ python vm_stress_script.py -i=1 -t=30 -r=15 -mem=0.25
Virtual memory status:
        Total: 7.0 GiB
        Available: 2.0 GiB
        Used: 5.0 GiB
Starting stress-ng:
        Stress workers: 1
        Allocated memory: 647M (25.0%)
        Methods: all
        Time per test: 30s
        Total time: 30s
Running 1 of 1 iterations...
```

Figure 5.3: Running the Python script

[5]Included as a separate file alongside this thesis.

[6]This parameters has a special non-numeric value — *dynamic*. It is used to trick Hyper-V and work around a particular dynamic memory limitation which will be discussed later in this thesis.

Data gathering

A virtual environment was created in VMware Workstation 14 Player to host a data gathering and management server. The virtual machine ran a preconfigured Zabbix Appliance (Zabbix 3.4.7 (20 February 2018)) operating on Ubuntu 16.04. It is a popular open-source monitoring solution to help monitor, maintain and collect data from hypervisors and VMs. Data gathering could have been done individually on every hypervisor, but the level of monitoring features vary drastically per hypervisor.

In addition to Zabbix, a Unix-based *collectd* daemon was set up on KVM. Collectd is a system and application performance metrics software which periodically collects and stores data in RRD format. Reason for collectd is simple: out-of-the-box Zabbix has its limitations, and it is very challenging to find pre-made templates to collect necessary information from KVM. It is possible to build custom templates, but that would require a substantial investment of one's time and effort to make it worthwhile. Collectd has all the necessary tools already built-in (e.g., virtual machine monitoring) and in combination with *Collectd Graph Panel* it displayed the data as easy to interpret graphs.

Other tools

Remmina[7] is a free remote desktop client that supports multiple network protocols (RDP, VNC, SSH and so forth) which makes it a versatile tool for system administrators and hobbyists alike. Remmina served as the primary tool to manage Hyper-V 2012 as every tool developed by Microsoft did not comply with Linux.

Virtual Machine Manager[8] developed by Red Hat served as Xen's administrative panel in addition to command-line based *xl tools stack* [40].

VMware ESXi and oVirt (KVM) ship with dedicated web-based administration panels and no additional tools were required.

[7]Remmina v1.1.2.
[8]Virtual Machine Manager v1.4.0.

6. Analysis

6.1 VMware ESXi

6.1.1 Memory optimization

Zabbix gathered the stress results and automatically calculated average values for every optimization tool expect for ballooning. Balloon data is based on the highest measured value. Inflated balloons did not stay active for very long — average inflation span varied from 3-5 minutes per iteration, meaning that over the course of a 40-minute test scenario averaged balloon data would have resulted in unrealistically low values. Note that whenever ballooning is mentioned it is nothing more than the amount of memory relcaimed by the host machine. Zabbix's results were used to plot the following bar charts (see 6.1). The vertical axis represents the amount of reclaimed memory by the optimization tools Virtual machines are positioned on the horizontal axis. To make charts as compact as possible shorter names were generated for operating systems.

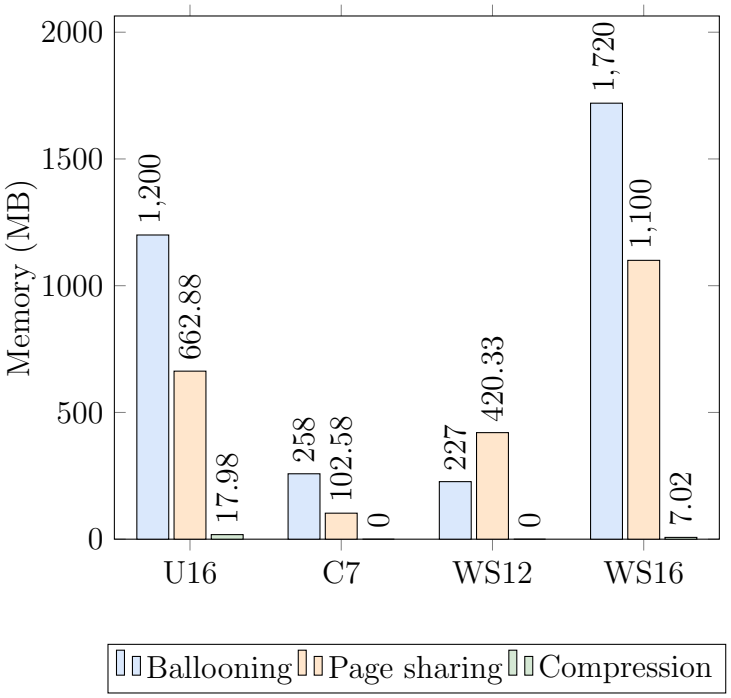

Figure 6.1: VMware — Memory optimization enabled

- U16 - Ubuntu 16.04
- C7 - CentOS 7
- W12 - Windows Server 2012 R2
- W16 - Windows Server 2016

VMware's first test condition (figure 6.1) had all the optimization tools (ballooning, page sharing, compression) enabled. From Ubuntu ballooning successfully retrieved 1.2 GB of RAM. Windows Server 2012 R2 and Windows Server 2016 were forced to return 942.2 MB and 1.72 GB of memory respectively. CentOS was the most resilient system delivering only 258 MB to the host. Page sharing was most effective in Windows OSs which is not that unusual considering that these two systems are closely linked. From Windows

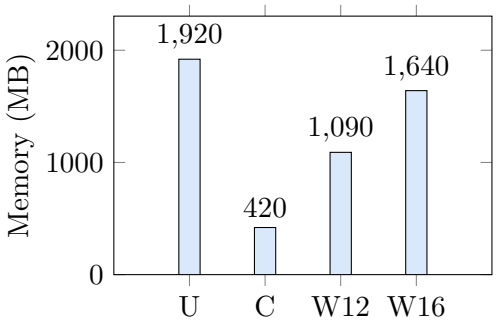

Figure 6.2: VMware — Ballooning only

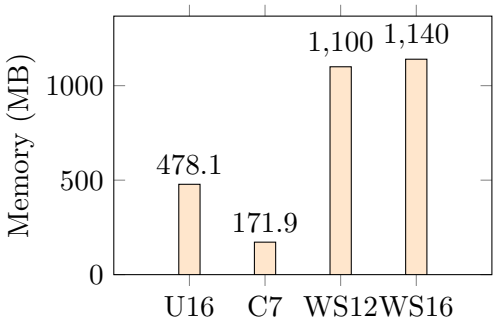

Figure 6.3: VMware — Page sharing only

Server 2012 R2 it reclaimed 420.33 MB and from Windows Server 2016 1.1 GB of RAM. In Ubuntu, page sharing managed to save 662.88 MB of memory. Once again CentOS was the most resistant with page sharing saving 102.58 MB of memory. Compression was pretty much non-existent all across. Only measurements were made in Ubuntu (17.98 MB) and Windows Server 2016 (7.02 MB).

During the second test condition (figure 6.2), only one memory reclamation tool was left running — ballooning. Ubuntu and Windows Server 2016 were the most willing to return memory to the host. Ubuntu gave up 1.92 GB, and Windows Server 2016 released 1.64 GB. Windows Server 2012 R2 did not fall far behind the previous two: 1.09 GB. CentOS returned 420 MB of RAM.

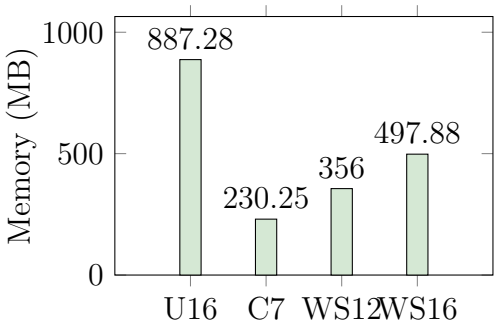

Figure 6.4: VMware — Compression only

As stated few paragraphs back Windows Server 2012 R2 and Windows Server 2016 have much in common in their overall architecture. Therefore it is no surprise that page sharing was most productive (figure 6.3) on Windows OSs. Windows Server 2016 returned 1.14 GB and Windows Server 1.1 GB of random access memory. Over at Linux page sharing managed to relieve 478.1 MB (Ubuntu) and 171.9 MB (CentOS) of RAM.

If one looks back at the first test condition (figure 6.1), then it is obvious that compression was the least effective or at least it did not get called out as much compared to the other tools. Surprisingly once compression was left on as a sole optimization (figure 6.4) method it managed reclaim 887.28 MB of RAM in Ubuntu, 230.25 MB in CentOS, 356 MB in Windows Server 2012 R2 and 497.88 MB in Windows Server 2016.

For a better comparison between VMware's various memory optimization and reclamation tools the following chart was generated (figure 6.5). Y-axis portrays the amount of reclaimed memory, and the X-axis denotes memory optimization tools where each bar represents the average memory reclaimed by the previously discussed stress conditions. For example across four VMs in the first test condition, an average of $(1200 + 662.88 + 17.98 + 258 + 102.58 + 227 + 420.33 + 1720 + 1100 + 7.02)/4 =$ **1428.95 MB** was reclaimed

by ballooning, page sharing, and compression combined. With only ballooning enabled this amount was 1267.5 MB. Page sharing followed with 722.5 MB and last, but not least, compression with 492.85 MB.

Based on this data it can be reasoned that out of ESXi's memory optimization and reclamation tools ballooning was the most capable of retrieving memory with page sharing and compression taking a smaller role. Of course, that does not mean that the last two should be rendered useless — each tool managed to do its job, and best results were accomplished as every optimizer was enabled and active.

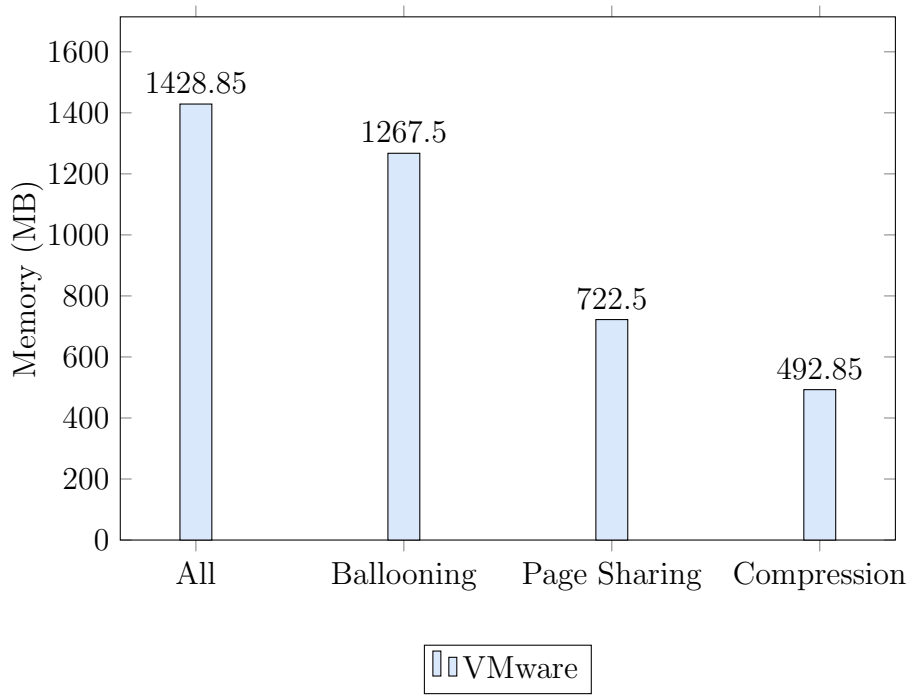

Figure 6.5: VMware — Memory reclamation by opt.method (avg)

6.1.2 Performance

In this subsection, a subjective assessment on VMware's hypervisor and VMs is given. Subjective meaning that no specific benchmark tests were run nor any data gathered — systems were simply tested by hand before, during and after tests had completed.

ESXi successfully survived all the test conditions: memory optimization enabled, disabled and one optimization tool at a time. Even during the extreme memory pressure without any reclamation tools at hand, the host kept itself operational — administrative panel was fully accessible and responsive. Warnings informed that the system was in low memory state, but that was the extent of it.

Virtual machines, on the other hand, did not do as well. As stress tools were running guests became extremely slow and at times even unresponsive. Once stress testing had completed, however, guests gradually returned to the optimal state. Worst outcome was during during the second test condition (no optimization) that ended in Windows Server

2012 R2, and Windows Server 2016 being forcefully shut down as they froze. Linux operating systems managed to shut down on their own barely.

6.2 Hyper-V

Compared to other hypervisor vendors Hyper-V had the most straightforward and minimalistic user interface. Hyper-V is aiming at ease of use, fast setup, and configuration process. However, it only supports one memory reclamation tool — a variation of ballooning which Microsoft itself identifies as *dynamic memory* [10]. Dynamic memory has three main components that can be modified.

- *Startup RAM* - the amount of memory allocated to the VM as it boots
- *Minimum RAM* - the minimum amount of memory guaranteed to a VM
- *Maximum RAM* - the total amount of memory that can be allocated by the hypervisor

Dynamic memory has an inherent flaw — guests are not aware of the maximum amount of random access memory potentially available to them. Take for example a VM with following parameters: startup RAM of 1024 MB; minimum RAM of 1024 MB; maximum RAM of 4096 MB.

Once the guest has booted, and the user starts a resource manager (i.e., Window's Task Manager; Linux's htop) it will display 1024 MB as the total amount of RAM, not 4096 MB. If this VM comes under memory pressure, Hyper-V will dynamically allocate more memory — a process that can be observed from any resource manager in real time. This design comes with its intricacies. Applications that perform memory checks before installation or startup would fail, because, from the guest's point of view, there is not enough memory available which might or might not be accurate [12]. This exact predicament occurred during stress tests conducted as part

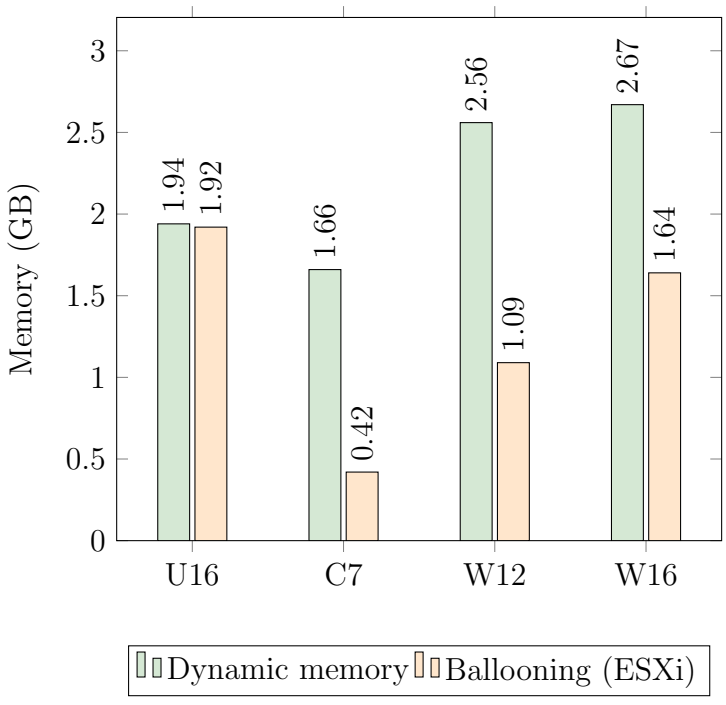

Figure 6.6: Hyper-V — Dynamic memory

of this thesis. Linux guests failed to start the stress tool — not enough memory. A workaround was devised, and thankfully the solution was rather easy — start *stress-ng* with more than one worker, allocating enough memory at the start to trigger Hyper-V's dynamic memory. As it turns out this is a legitimate problem in the production environments too. One workaround[1] is to use MS Paint (only works in Windows VMs), expand its canvas size to the absolute maximum. This operation will consume a lot of resources and forces dynamic memory to hot-add memory — allowing to install and start programs that previously failed to do so [9].

6.2.1 Memory optimization

Dynamic memory is a very aggressive memory reclamation method. If a virtual machine is not fully utilizing resources that have been allocated to it, then it takes dynamic memory a minute, and in some cases thirty (30) seconds, or less to reclaim it. It is apparent as one looks at the figure 6.6. In each VM, dynamic memory managed to retrieve more than 1 GB of RAM — Ubuntu 1.94 GB; CentOS 1.66 GB; Windows Server 2012 R2 2.56 GB and Windows Server 2016 2.67 GB — exceeding VMware.

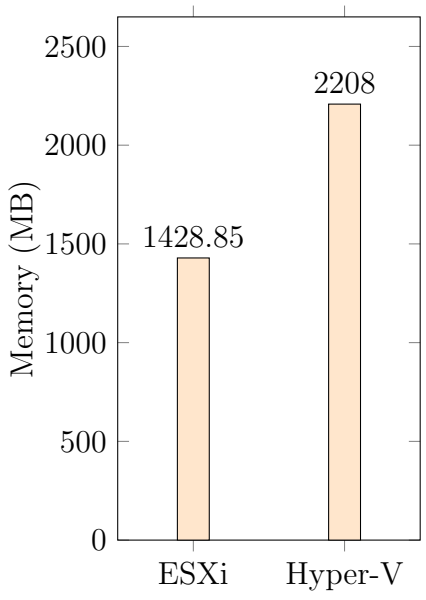

Figure 6.7: ESXi and Hyper-V

A pattern is starting to develop — Hyper-V's bar chart resembles those of VMware (figure 6.2). Both Windows Server guests were most willing to give up memory as the host aimed to reclaim it. Ubuntu came to a close third and CentOS being the most resilient like before. It is too early to decide whether this patternicity was caused by the stress tools or guest operating systems.

Comparing Hyper-V's dynamic memory to VMware's ballooning chart (figure 6.6) it is ever more apparent that Hyper-V had better results. From Ubuntu Hyper-V and ESXi managed to reclaim about the same amount of RAM (1.93 GB). In CentOS, VMware did a lesser job and retrieved four times less (400 MB) compared to Hyper-V's 1.66 GB. In Windows OSs, both hypervisors managed to reclaim more than 1 GB of memory, but Hyper-V surpassed VMware by an extra gigabyte.

[1]Recommended by Microsoft.

6.2.2 Performance

Hyper-V's and its VMs performance evaluations are subjective, as they were with VMware. Hyper-V survived both stress conditions and did not crash. The system remained online and responsive even under extreme memory pressure caused by disabling dynamic memory — only warning about the low memory state. The probability that Hyper-V[2] was ever going to crash during these test conditions was extremely low. Hypervisors reserve a certain amount of physical resources to themselves, and virtual machines do not have any access to that domain making it very improbable for VMs to crash the host.

Linux guests were sluggish compared to Windows Servers even after implementing all the configurations recommended by Microsoft. Other than that both Linux and Windows OSs displayed slowness and unresponsiveness during testing — just like in VMware. Once stress tools had finished and around 2-4 minutes had passed virtual machines stabilized.

6.3 KVM

6.3.1 Memory optimization

Collectd gathered test results from KVM. Unfortunately one of the plugins did not operate as expected — kernel same page merging measurement script. Different configurations based on tips and tricks from the Internet were attempted but failed to rectify the problem. That rendered KSM's data gathering from VMs impossible.

Fortuitously, data was also gathered from the hypervisor. As KVM only supports two memory reclamation tools it was possible to extract[3] information linked to page sharing from KVM's hypervisor memory dataset. Reverse engineered data was not as accurate with no indication how much sharing took place in virtual machines individually, but at least it gave a rough estimate how well this memory optimization tool performed.

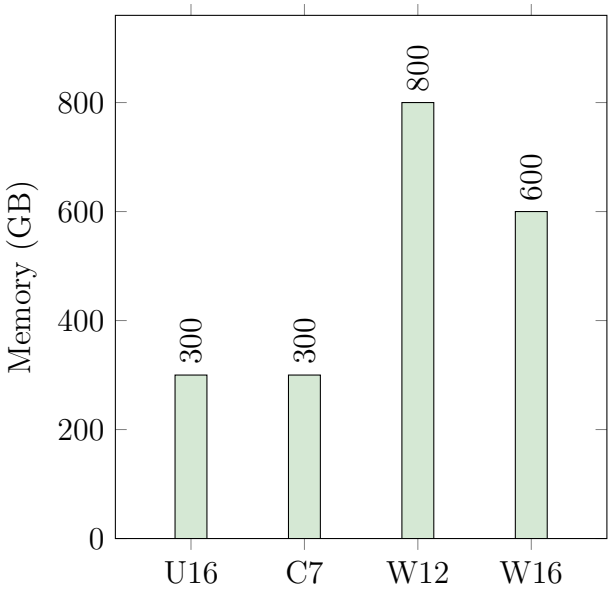

Figure 6.8: KVM — Ballooning

With both memory optimization tools enabled KVM managed to reclaim 2 GB by ballooning (figure 6.8). Ubuntu and CentOS recovered roughly 300 MB of RAM each. Windows Server 2012 R2 and Windows Server 2016 800 MB and 600 MB respectively. KVM's ballooning chart resem-

[2]Or any other hypervisor for that matter.

[3]Reverse engineer.

bles Hyper-V's dynamic memory results — both Linux OSs freed around the same amount of memory with Windows Servers leading ahead with double the memory.

Kernel same page merging managed to reclaim an average of 650 MB across four VMs approximately. It must be remarked once again that KSM data gathering in VMs failed and there is no data to display reclaimed memory proportionality per guest. On the other hand, based on the data analyzed in previous sections, it does not seem a far cry that KSM chart would look similar to VMware's page sharing (figure 6.3). This proposal is also backed by KVM's ballooning which indicated that Linux OSs are less prone to give back memory next to Windows operating systems — a trend seen across multiple hypervisors. In the end, there was no hard data, and this assumption should be taken with a pinch of salt.

6.3.2 Performance

KVM as a hypervisor was as sturdy as ESXi and Hyper-V — under extreme memory pressure the administrative panel was reachable and responsive. As with the previous two hypervisors, KVM also warned as the system was on the verge of running out of resources, but since it has reserved a specific amount for its own use then these warnings are more related to virtual machines, and it should not cause any serious problems in the hypervisor.

Virtual machines were more responsive compared to Hyper-V's guests. Guest tools installation was considerably more difficult and not as straightforward as with VMware and Microsoft. In low memory state guests became less responsive, but none of them ever crashed. Once the testing was complete machines returned to their optimal state.

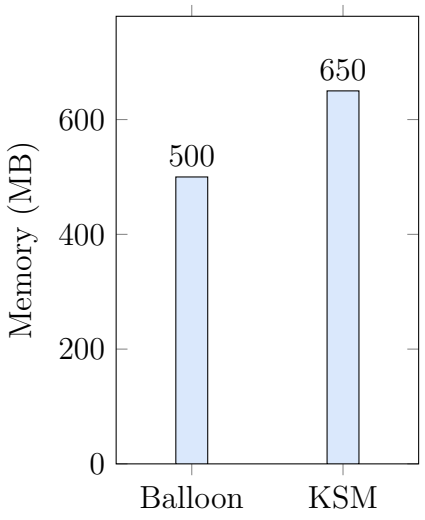

Figure 6.9: Ballooning and KSM

6.4 Xen

As stated few chapters back, Xen hypervisor failed to meet specific criteria — automated memory management. That made it difficult to test and compare it to other hypervisors. Xen is available as an open-source and a commercial product (Citrix), and both versions were given a go.

The installation of the latest Citrix XenServer 7 failed as it continually froze during the installation[4] and none of the recommendations on the community forum and by Citrix support did not resolve the problem. Out of interest, an older Citrix XenServer 6.2 was

[4]https://support.citrix.com/article/CTX136517

installed instead — successfully. That lead to the conclusion that Citrix XenServer 7 did not comply with Sun Fire X2250. Unfortunately, it was not possible to request a trial license for such an old hypervisor, rendering Citrix XenServer's performance testing impossible.

After that, the open-source variant was given a try instead. Installation was a success, and the hypervisor was populated with VMs. Unfortunately, it soon came to attention that although Xen supports memory optimization; that process is entirely manual and the only way to get access to advanced memory optimization features is to use a dedicated frontend, i.e., Citrix XenServer as alternative open-source solutions are outdated and very cumbersome to set up on modern operating systems.

Summary

In conclusion chapter two through four served as the theoretical background. Chapter two introduced the concept of virtualization and laid out its history in a very brief manner. In the third chapter two vital components of virtualization were presented — hypervisors and virtual machines. Chapter three revised the concept of virtual memory; its connection to hypervisors and virtual machines and what sort of memory optimization and reclamation tools are available today. In the fifth chapter experiment methodology and the laboratory set up process was described in detail.

Based solely on the numerical results Windows Server 2012 R2 with Hyper-V was able to reclaim most memory from its VMs with VMware ESXi taking the second and Kernel-based Virtual Machines third place, but Hyper-V has an inherent feature that makes its deployment and reliability in a production environment somewhat questionable. As it was pointed out in the thesis VMs running on top of Hyper-V are not aware of the maximum random access memory they have the potential access too. It took a bit of trickery to work around this limitation and force Hyper-V to allocate the much-needed RAM. On the other hand, once dynamic memory got triggered it acted quickly in both allocating and reallocating memory. If a VM did not utilize available memory, it took Hyper-V less than a minute or even less to reclaim it. The hypervisor never crashed nor did the VMs, but compared to ESXi and KVM guests Hyper-V's VMs were more sluggish and less responsive.

Although according to the data VMware ESXi was the second hypervisor based on the amount of reclaimed memory, it is still the most mature package out of the three. Easy to install and configure and it employs three memory optimization and reclamation tools[5] — offering a wide range of possibilities. Keep it in mind that tests conducted throughout this thesis were mostly based on default system parameters, but in a production environment, a more specific approach route might be taken. In those situations, it is beneficial to have a tool that offers more than one resolution to the same problem and provides the system engineer with options to configure virtualization environments based on specific requirements. If customization is a must, then VMware ESXi is a viable option.

Performance wise Kernel-based virtual machines were the poorest, but these results should not be taken as absolute truth, and future research is hugely commendable. As it was mentioned in the KVM's analysis the data gathering software did not operate as expected and some of the data was derived from other related datasets. The hypervisor did not crash during the test nor did the VMs — an excellent result in any hypervisor under extreme memory pressure. Once the tests were done, the system returned to a stable state.

KVM, depending on the frontend, can offer an experience similar to VMware ESXi — it is a hypervisor with customization and large production environments in mind. Guest drivers installation was not as straightforward as with ESXi, but the presence of drivers did make a difference in performance.

[5] Four if including memory overcommitting.

Although KVM performed poorly, it holds great potential. Based on the article by Trevor Pott *The Real Threat to VMware: KVM* (2016), it is seen as one of the possible future contenders to VMware and that in itself is a noble thing. VMware is the current market leader (70%) in the hypervisor's industry, and that can create a state where innovation stops. If KVM manages to grab a significant portion of the market, and it seems to have the potential to do so, it would stir up the virtualization world and could lead to innovations — and that would be beneficial to both virtualization as a technology and industries that rely on it.

This thesis set out to test and analyze memory optimization and reclamation tools in widely used hypervisors (VMware ESXi, Microsoft Hyper-V, Kernel-based Virtual Machine, Xen). The aim was to set up a dedicated environment and stress test both the hypervisors and virtual machines within them to see how much memory could be reclaimed and optimized to handle resources as efficiently as possible. Overall the testing and analysis could be considered a success, even when taking into account the numerous problems and hiccups along the way. That is a legitimate part of scientific research — things do not always work, and that in itself is a valuable lesson.

Acknowledgements

I would like to thank my supervisor, Meelis Roos, for helping me to choose a topic for my graduation thesis, providing hardware and guiding me through the creative process.

License

I, Tambet Viitkar

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   Hypervisor and Virtual Machine Memory Optimization Analysis

   supervised by Meelis Roos.

   (a) reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   (b) make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **20.05.2018**

References

[1] Ishan Banerjee, Fei Guo, and Kiran Tati. *Memory Overcommitment in the ESX Server*. 2013. URL: http://bit.ly/2GwbMfx (visited on 03/17/2018).

[2] Ed Carrel. *What's the difference between operating system swap and page?* Nov. 6, 2009. URL: https://stackoverflow.com/questions/1688962/whats-the-difference-between-operating-system-swap-and-page (visited on 04/20/2018).

[3] CentOS. *What is CentOS Linux?* 2018. URL: https://wiki.centos.org/ (visited on 04/04/2018).

[4] Charu Chaubal. *The Architecture of VMware ESXi*. Oct. 24, 2008. URL: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/ESXi_architecture.pdf (visited on 04/09/2018).

[5] Cnet. *Sun Fire X2250 Xeon E5405 2 GHz*. 2018. URL: https://cnet.co/2IorgTi (visited on 04/03/2018).

[6] Sarah Cooley. *Introduction to Hyper-V on Windows 10*. Apr. 7, 2018. URL: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/ (visited on 04/12/2018).

[7] Sarah Cooley, Justin Terry, and Hannah Juarez. *Hyper-V Architecture*. Jan. 11, 2018. URL: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture (visited on 04/09/2018).

[8] A. Finn. *How Much Memory Does My Hyper-V Host Require?* Sept. 30, 2015. URL: http://www.aidanfinn.com/?p=19043 (visited on 04/17/2018).

[9] A. Finn. *Software Setup Does Not Meet Memory Requirements with Dynamic Memory Enabled*. Dec. 20, 2010. URL: http://www.aidanfinn.com/?p=11007 (visited on 04/17/2018).

[10] Aidan Finn. *What is Hyper-V Dynamic Memory?* Mar. 31, 2014. URL: https://www.petri.com/hyper-v-dynamic-memory-overview (visited on 04/12/2018).

[11] Vivek Gite. *How To Stress Test CPU and Memory (VM) On a Linux and Unix With Stress-ng*. Jan. 30, 2015. URL: http://bit.ly/2HTTzN5 (visited on 03/12/2018).

[12] Eric Horschman. *Hypervisor Memory Management Done Right*. Feb. 18, 2011. URL: https://blogs.vmware.com/virtualreality/2011/02/hypervisor-memory-management-done-right.html (visited on 04/12/2018).

[13] Intel. *Intel Xeon Processor E5405*. 2007. URL: https://intel.ly/2ItbK93 (visited on 03/29/2018).

[14] M. Tim Jones. *Application virtualization, past and future*. 2011. URL: https://ibm.co/2H8WDlH (visited on 03/03/2018).

[15] M. Tim Jones. *Virtualization for Embedded Systems*. Apr. 19, 2011. URL: https://www.ibm.com/developerworks/library/l-embedded-virtualization/ (visited on 04/12/2018).

[16] Kernel.org. *Swap Management*. 2018. URL: https://www.kernel.org/doc/gorman/html/understand/understand014.html (visited on 04/20/2018).

[17] Sean Michael Kerner. *How Did KVM Virtualization Get Into the Linux Kernel?* Nov. 18, 2013. URL: http://www.eweek.com/cloud/how-did-kvm-virtualization-get-into-the-linux-kernel (visited on 04/20/2018).

[18] Colin Ian King. *Stress-ng*. Mar. 12, 2018. URL: http://kernel.ubuntu.com/~cking/stress-ng/ (visited on 03/12/2018).

[19] Colin Ian King. *Stress-ng Manual*. Feb. 22, 2018. URL: http://kernel.ubuntu.com/~cking/stress-ng/stress-ng.pdf (visited on 03/14/2018).

[20] Anis Koubaa. *A Service-Oriented Architecture for Virtualizing Robots in Robot-as-a-Service Clouds*. 2014. URL: http://bit.ly/2jdpZUD (visited on 04/27/2018).

[21] KVM. *Automatic Ballooning*. Jan. 23, 2014. URL: https://www.linux-kvm.org/page/Projects/auto-ballooning (visited on 04/10/2018).

[22] KVM. *Management Tools*. Apr. 12, 2017. URL: https://www.linux-kvm.org/page/Management_Tools (visited on 03/10/2018).

[23] OpenSUSE. *Introduction to KVM Virtualization*. 2018. URL: https://doc.opensuse.org/documentation/leap/virtualization/html/book.virt/cha.kvm.intro.html (visited on 04/21/2018).

[24] oVirt. *What is oVirt?* 2018. URL: https://www.ovirt.org/documentation/introduction/what-is-ovirt/ (visited on 04/10/2018).

[25] Matthew Portnoy. *Virtualization Essentials (Second Edition)*. Sybex, 2016. ISBN: 978-1-119-26772-0. URL: https://www.amazon.com/Virtualization-Essentials-Matthew-Portnoy/dp/1119267722?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1119267722.

[26] Trevor Pott. *The Real Threat to VMware: KVM*. July 19, 2017. URL: https://virtualizationreview.com/articles/2016/07/19/the-real-threat-to-vmware-is-kvm.aspx (visited on 04/22/2018).

[27] RedHat. *What is KMV?* 2018. URL: https://www.redhat.com/en/topics/virtualization/what-is-KVM (visited on 04/20/2018).

[28] RedHat. *What is virtualization?* 2018. URL: https://red.ht/2HdgGiZ (visited on 03/03/2018).

[29] RoboEarth. *What is Cloud Robotics?* Apr. 27, 2018. URL: http://roboearth.ethz.ch/cloud_robotics/index.html (visited on 04/27/2018).

[30] Jonathan Strickland. *How Server Virtualization Works*. 2018. URL: http://bit.ly/2EhX7mk (visited on 03/03/2018).

[31] Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems (4th Edition)*. Pearson, 2014. ISBN: 978-0-13-359162-0. URL: https://www.amazon.com/Modern-Operating-Systems-Andrew-Tanenbaum/dp/013359162X?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=013359162X.

[32] UKEssays. *The History Of Virtualization*. 2015. URL: http://bit.ly/2q4gBqq (visited on 03/03/2018).

[33] VMWare. *Virtualization Essentials*. 2018. URL: http://bit.ly/2GQsklR (visited on 03/03/2018).

[34] VMware. *Understanding Memory Overhead*. URL: https://pubs.vmware.com/vsphere-4-esx-vcenter/index.jsp#com.vmware.vsphere.resourcemanagement.doc_40/managing_memory_resources/c_understanding_memory_overhead.html (visited on 04/16/2018).

[35] VMware. *Understanding Memory Resource Management in VMware ESX 4.1*. 2010. URL: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/vsp_41_perf_memory_resource-management-white-paper.pdf (visited on 04/05/2018).

[36] VMware. *Understanding Memory Resource Management in VMware ESX Server*. 2009. URL: http://bit.ly/2q4uR2o (visited on 03/16/2018).

[37] Ubuntu Wiki. *Ubuntu - List of Releases. List of Releases*. 2018. URL: https://wiki.ubuntu.com/Releases (visited on 04/04/2018).

[38] Wikibooks. *Introduction to Information Technology — Virtualization*. Ed. by Wikibooks. 2016. URL: http://bit.ly/2Ha5SSy (visited on 03/03/2018).

[39] Wikipedia. *Ubuntu (operating system)*. 2018. URL: https://en.wikipedia.org/wiki/Ubuntu_(operating_system) (visited on 04/04/2018).

[40] Wikipedia. *Virtual Machine Manager*. 2018. URL: http://bit.ly/2uQxF88 (visited on 04/03/2018).

[41] Wikipedia. *Windows Server 2012*. 2018. URL: https://en.wikipedia.org/wiki/Windows_Server_2012 (visited on 04/04/2018).

[42] Wikipedia. *Windows Server 2016*. 2018. URL: https://en.wikipedia.org/wiki/Windows_Server_2016 (visited on 04/04/2018).

[43] Wikpedia. *CentOS*. 2018. URL: https://en.wikipedia.org/wiki/CentOS (visited on 04/04/2018).

[44] Matt Woodward. *Swap vs. Paging File*. 2018. URL: https://archive.techarp.com/showarticle1776.html?artno=143&pgno=1 (visited on 04/20/2018).

[45] XenProject. *Best Practices*. 2015. URL: https://wiki.xenproject.org/wiki/Xen_Project_Best_Practices (visited on 04/08/2018).

[46] XenProject. *History*. 2014. URL: https://www.xenproject.org/about/history.html (visited on 04/08/2018).

Appendix

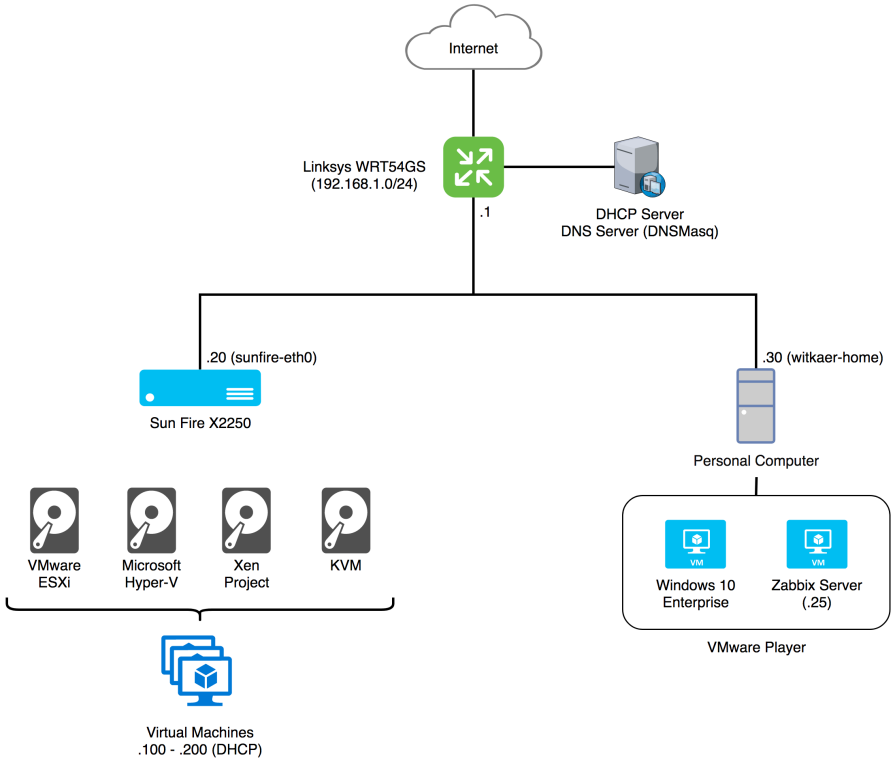

Figure 6.10: Network topology