Master Thesis

The Virtual Machine Delivery Network

University of Twente
Computer Science
Netherlands Organisation for Applied Scientific Research (TNO)

Maarten Fonville
2014

Supervisors:
Dr. Ir. Oskar van Deventer
Dr. Ir. Pieter-Tjerk de Boer
Dr. Ir. Marten van Sinderen

Abstract

Distributing virtual machine images over the internet via traditional methods wastes network resources. A proposed solution to this problem is the Virtual Machine Delivery Network (VMDN). A VMDN optimizes the distribution of virtual machine images using concepts from content delivery networks combined with knowledge that exists about the inner structure of virtual machines and their disk images. In this research this virtual machine image's disk structure has been analyzed which resulted in three different type of data structures. For each of these data structures a different strategy has been developed for the VMDN prototype implementation. Three different scenarios were created to get a set of possible performance factors. The strategies' performance was measured using these performance factors and the applicability of each strategy per scenario has been evaluated. The findings of the study are that a VMDN can be much more efficient bandwidth efficient than traditional distribution methods, but that the most effective strategy differs greatly per given context.

Table of Contents

1 Introduction

Virtual machines; a term that you might not be acquainted with, but very important in today's digital world [1], [2]. The virtual machine is an software-abstraction of a real computer and is an ideal unit for computing; we will explain you why. Probably you have a desktop computer at your work, to perform your job's activities. The computer hardware is bought, software applications were installed and software licenses for it were obtained. This computer is 24 hours a day at your desk, but you only work 8 hours a day. Meanwhile computer hardware is precious and expensive. So only using the hardware for a third of the time makes introduces unnecessary costs. It could be compared to paying the whole year round for a holiday home that is only used on the weekends. For the holiday home a solution to this problem is possible through the use of time-sharing. The user only pays for the time the holiday home is actually being used. When it is not used and the holiday home would be idle, it is rented out to another user. By using such a scheme, the costs of each user can be greatly reduced by dividing the costs of the home-ownership over multiple users.
For computing time-sharing would also be possible, you could choose to rent out your computer resources to another party at the time your machine would be idle. But a consequence would be, just like with the equipment of the holiday home, that you'd have to share the contents of the computer with the other users. Sharing the contents of your computer with other users could pose security, legal and many other possible problems.

Virtual machines are a solution to allow the sharing of computing resources without having to share computer contents with other users. Virtual machines are possible by creating an abstraction layer between the physical computer hardware that provides the computing resources and the computer components that are visible to the software [3]. By introducing this abstraction, the physical computing resources can be shared while the computers and their contents are isolated. It also allows one physical computer to serve multiple virtual machines at the same time, dividing its resources.

This abstraction allows a new cost model for computing resources: only having to pay for the computing resources that are actually used, instead of paying the high costs associated

with the ownership of a fixed amount of computing resources. This payment scheme enables parties to greatly reduce their costs if their demand of computing resources varies greatly over time. Think e.g. about video-on-demand providers like Netflix [4]. During the day there is a low demand for computing power, because people are don't have any free time to watch a movie. But in the evening a lot of customers switch on their televisions and want to watch a movie from Netflix. Obviously there are times of the day when there is a peak-demand for computing-infrastructure for the video-provider, while at other moments demand is low. They greatly benefit from a pay-as-you-go scheme for computer resources.

Thus virtual machines are a proper answer to meet the demands for flexible computer resources. Their large advantage is that these virtual machines are able to provide the exact same functionality of real physical computers. But instead of directly executing the software's hardware instructions on physical components, they are interpreted by a virtual machine monitor (also known as a hypervisor) [5]. The hypervisor then executes its interpreted instructions from the virtual machine on the real physical computer.
One of the great advantages of the virtual machines is that they can be created when there is demand for their existence and be removed when there is not. Also these virtual machines can be moved from one physical machine to another. By optimally combining multiple virtual machines on physical machines, matching resources to actual computing demand, less physical computing resources are necessary in the end. This makes the virtual machine one of the most practical '*units*' to work with in the world of computing where flexibility is key. And we foresee a future where this unit will become more and more important throughout time.

In daily live we are already acquainted with the fact that a physical computer has a location as one of its properties. The computer has to be placed somewhere, it cannot be nowhere. This location can influence the usability and functionality of the computer. E.g. if you want to use your laptop computer to process some text of your USB stick in your text editor, you need the screen, keyboard and USB-port to be accessible.
If your laptop would be inaccessible, for example being in another room, the laptop could not provide you with the means to fulfill your goal of text-editing. You'd have to move the laptop first to your room, before you would be able to use it in a purposeful manner.

The same applies for virtual machines, also for them their location is a property and it can influence their functionality. And just like laptop computers, virtual machines can be moved. But unlike physical computers, where you are moving the physical components from one room toe the other. With virtual machines, you move virtual components. So what do these virtual components embody? The components are, as described before, just software interpretations done by the hypervisor. But the hypervisor needs a list of which components and how they are configured and which data they contain to be able to recreate these virtual components. This list of components is stored per virtual machine as a virtual machine configuration file. Most of the components of the virtual machine barely have any data which (permanently) resides inside them. But one component is the exception: the hard disk. The hard disk contains many bytes of data and to let the hypervisor recreate the same hard disk all these bytes have to be stored and transferred if

the virtual machine is moved. This can be a challenge when a virtual machine with a large disk has to be moved, or if a virtual machine has to be moved over a large distance. Because this will imply a large amount of network resources would have to be used to transfer all these bytes to the target location.

Within this research we will introduce a solution for optimizing the moving of virtual machines over computer networks, using as little resources as possible to get virtual machines from a source to a target location.

In chapter 2 the motivation for this research will be described. The problem with current distribution of virtual machines is explained and the role for the proposed Virtual Machine Delivery Network (VMDN) is to solve this problem

In chapter 3 the technical details of current virtual machines and Content Delivery Networks (CDNs) are described. This background also shows why regular CDNs are currently not sufficient for solving the VM distribution problem.

In chapter 4 three example situations for a virtual machines to be distributed are introduced. These three examples will be used as possible scenarios within this research. For each of these scenarios performance factors that could be considered relevant for their distribution and deployment scenario are highlighted and discussed.

In chapter 5 the state of the art is discussed. In the first subchapters the current virtual machine image structure is derived from findings in literature. With this information more related work for each type of data-object that is found in the virtual machine's image structure is discussed.

In chapter 6 the proposed VMDN is introduced. The VMDN consists of a prototype using three different strategies. These three strategies all have a different technical background, design and implementation. Per strategy these aspects will be discussed. A short analysis will be performed on the relative performance scores between the strategies.

In chapter 7 the possible future work is discussed. Focus within the future work is improvements on the current prototype and on the so-called "YouVM" example. A parallel for future virtual machine distribution as YouTube did for video distribution.

In chapter 8 a short summary of the conclusions of the research can be found.
Also a glossary has been included for the reader, where some of the used tools and non-generic abbreviations are indexed.

In the appendix A a more detailed analysis is performed. First per strategy the strengths and weaknesses are discussed and rated. Using this information these strengths and weaknesses are analyzed per applicability on the in chapter 4 introduced scenarios. These results are summarized into a scoring-table to show which strategies are viable (or not) per scenario.

2 Rationale

As described in the introduction is in the world of information technology (IT) services for certain scenarios, like Netflix, a demand for flexibility of computing resources. A new business model was an important part of the solution of IT resource providers to offer their customers this demanded flexibility: the 'as-a-Service' paradigm [6].
This new paradigm is popular and has resulted in an enormous growth of the so-called 'cloud computing' industry, which offers their IT services within this as-a-Service paradigm [7].
But these as-a-Service cloud computing applications did also issue new demands on the underlying computing infrastructure. To be able to offer the flexibility that the cloud computing services demand, the computing infrastructure itself also had to become more flexible. A key element for fulfilling this need of flexibility is '*hardware virtualization'*. Hardware virtualization encompasses the possibility to instead of having one computing-environment on one physical computer, that you can have multiple virtualized computing-environments on a single physical computer. These virtualized computer-environments are also moveable between different physical computers. This allows virtualization to be an ideal method for offering the flexibility necessary for cloud computing, because it eliminates the need for a physical computer to offer a computing environment. Instead, per demand a '*virtual machine'* can be created that offers the same computer-environment experience as a physical one provides. This makes the virtual machine a practical unit for offering IT services in cloud computing and it is to be expected that its importance will grow even more throughout time.

Since cloud computing providers already use the virtual machines to provide their services, it is also a unit that they can offer to their cloud computing customers. This virtual machine unit is then also the most basic unit that is offered within the typical cloud computing service stack [8]. Offering a cloud computing service to customers using these virtual machine units is called Infrastructure-as-a-Service (IaaS). IaaS enables the cloud computing providers and their customers to use full machine-like computing environments, through the usage of virtual machines. These virtual machines can be created at many different locations in the world and expose their services to clients, like end-users, via networking interfaces, e.g. via the internet. The virtual machines encapsulate applications, services, and data, which are stored in virtual machine images. These virtual machine images are often saved as very large files on a physical computer. In principle, IaaS clouds give users flexibility in application and service deployment as clouds enable dynamic creation, migration, and use of virtual machines over data centers at many locations in the world.

2.1 Problem Statement

In the introduction was described that a computer's location can influence its functionality. Also virtual machines have a location: the physical computer (also known as the '*host*') that executes the hypervisor software that runs the virtual machine (also known as the '*guest*'). For various purposes there can be a demand for a virtual machine to be available at a certain target location. The physical host that is this target location then needs access to the virtual machine's disk image to be able to execute the virtual machine. If this virtual machine disk image is not available at the target location, but only available at a remote source location, access to this disk image should be gained somehow. The easiest way to gain this access is by just copying the virtual machine's disk image from the source location to the target location via a computer network like the internet. But even though virtual machine images can be copied bit by bit between these locations, in practice the distributing of such large files over the internet is inefficient and wastes network resources. Especially if multiple copies of a virtual machine would have to be copied to multiple target locations. The problem addressed in this thesis is how to create a V*irtual Machine Delivery Network* (VMDN) that facilitates the exchange of virtual machine images between different locations to be substantially more efficient than simply distributing the images via the internet using conventional methods. The problem and proposed VMDN are illustrated in Figure 1.

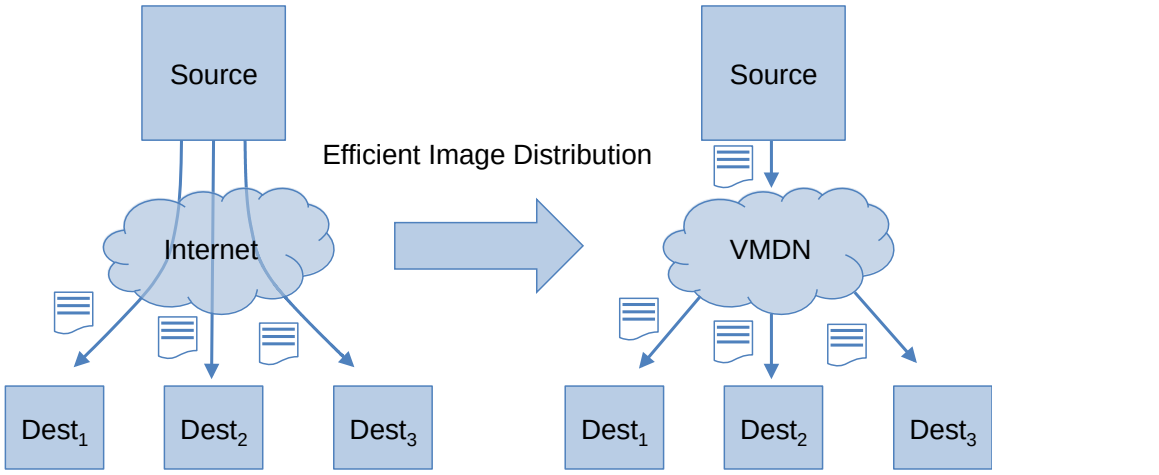

Figure 1 Distributing virtual machine images via the internet wastes network resources. A Virtual Machine Delivery Network (VMDN) optimizes the distribution of virtual machine images, possibly using concepts from content delivery networks and storage networks. The goals of this thesis are to identify methods for efficient image distribution, to quantify the efficiency of these methods, and to create a proof of concept demonstrating the workings of a virtual machine delivery network.

The presented problem is similar to distributing media content via the internet. For that problem a solution was found with the development of Content Delivery Networks (CDN) [9]. CDNs are networks of caches and media servers that are vastly distributed over different (geographical) locations all over the internet. CDNs enable more efficient delivery and higher performance of multimedia content to consumers. CDNs enable a better quality of experience to the consumers and offload the burden of delivering large quantities of data from the source providers of the media content. By using CDN infrastructure, the total amount of data that in the end has to be transferred over the various networks that make up the internet is reduced through the caching mechanisms.

Thus also the overall network itself benefits from the use of CDNs, by improving the available network capacity.

Unfortunately CDNs are not suitable for virtual machine distribution. CDNs rely on the property that content has to be a static and bitwise identical to be identified by the CDN infrastructure as a cacheable object. But unlike many other types of content, virtual machines are not static. With every execution their virtual machine image will change. Also the virtual machine's binary representation can greatly differ between virtual machine images that actually represent identical virtual machines. And though we know many virtual machines can be very alike, like having the same applications and files on their disk, their disk image's binary representation are nothing alike. Such overlap between different virtual machine image's cannot be discovered by current CDNs. This makes virtual machine's images to be really unsuitable objects to be cached in CDNs, because no cache-hits are to be expected given these circumstances. This while virtual machine images are very large in size and a lot of network resources could be saved if the distribution mechanisms would be improved.

Apparently there is a gap between the caching features of CDN that would be useful for Virtual Machine distribution and the constraints of current CDN implementations. That is why we propose a new concept to fill this gap: the VMDN (Virtual Machine Delivery Network). The VMDN should provide the same distribution-optimization features as a CDN does, but should be able to mitigate the current limitations of CDNs concerning virtual machines as content-type. Thus a VMDN should provide the methods to efficiently distribute virtual machine images from a source location to various target locations. The VMDN should leverage the knowledge that large virtual machine images do have a certain common structure: a structure that in the end represents a virtual machine. In this manner VMDNs should be able to use methods that are more advanced than those from traditional CDNs. This should enable VMDNs to implement effective caching and storage methods, reducing the amount of network resources necessary for distributing virtual machine images to the target locations.

Also it is important for the distribution of virtual machines that the concept of a copy has to be further questioned in the given context. A comparison could be made to other topics to answer the question "*what is a copy?*" Is a copy of a movie on VHS different from the original movie? And is the Blu-Ray version? Since all of these copies they are not a pixel-perfect copy of the original production. This is because they all use different technology and different lossy compression methods that produce small differences in the end-result. But nonetheless all of these copies are considered to be valid functional copies for the purpose of watching the movie.
The same questions could also apply to virtual machines. If copying a virtual machine, which factors are important for it to be perceived as a good, functioning copy? Which properties should it retain? Should it be (when comparing to movies a pixel-)perfect copy of the original? Or is a lossy-copy (that loses some of it details) with same functionality also sufficient?
These questions can also be an important factor for this research. Since making a copy that does not have to be completely bitwise identical to the original, but still equal enough

in functionality to be considered a copy could allow for more possible approaches in the VMDN distribution mechanism.

2.2 Research questions

The main research question for this research is as follows:

- Can an optimized Virtual Machine Delivery Network be substantially more bandwidth efficient for distributing virtual machines than using traditional Content Delivery Networks?

Sub-questions

To answer this question the following sub-questions were formulated:

1. What are the performance aspects concerning bandwidth efficient virtual machine distribution?
2. What is the bandwidth efficiency of the distribution of completely (bitwise) identical copies of virtual machines using traditional content distribution methods?
3. How can completely (bitwise) identical copies of virtual machines be more efficiently distributed by using a VMDN?
4. Are there more bandwidth efficient methods for the distribution of lossy copies virtual machines using a VMDN?

3 Background

3.1 Content Delivery Networks

As we described earlier, the current method for optimizing distribution of large or popular files is through the usage of CDNs. CDNs are run by companies that are specialized in delivering content for their customers (which is the party that provides the content) by mirroring it on various servers that are (geographically) spread out over the internet network [9], [10]. The benefits of using a CDN for the original content distributing party are a reduced load on the content providing party's servers, off-load of the burden at peak-time loads, increased performance and availability of the content to the end-users and a reduction in the amount of network capacity that is necessary and network-peering costs.

The business model of the companies running the CDNs relies on the fact that they can bundle the deliveries for the content of their customers all together. This allows them to reach an economy of scale that allows them to reduce the costs per 'unit'. A unit can be considered in this situation a delivery to an end-user for a piece of content that originally originates from one the customers. The economy of scale is also actually necessary to allow the service model of CDNs itself to function. They need many deliveries to reach the implicit geographical spread. Only by a having a great enough demand per geographical area the companies running the CDNs can afford to have servers available in all of those different regions to fulfill the content deliveries.

CDNs use statistical effects within the content's consumption, like Zipf's law and the Pareto principle, in conjunction with caching methods and content consumption prediction algorithms [11]–[14]. CDNs allows their customers' content to be replicated into their distribution network. CDNs can then mirror this content at different geographical locations and on different peering-level, depending on demand.
Each location where content is being mirrored has a limited set of storage available, so a choice has to be made where which content will be mirrored. The idea is to match the content of each cache in such a way that an optimal distribution is achieved. This optimum can be achieved if the amount of bytes that has to be transferred in total to handle all requests can be kept as low as possible for an affordable price.
If we combine this optimum with the fact that CDNs have to pay for each location, servers, storage and peering and have various options for each of these factors they have to make several choices when caching content. So when considering where to cache which content, the CDN has to make smart choices.
For content that is or large or is popular within a (single) distinct (network) region it is often smart to cache it as close as possible to the end-user (e.g. at the Tier 3 level, like the ISP's network) [15]. In that manner the highest amount of bytes that has to be transferred on the grand total can be saved.
Meanwhile, content that is smaller in size or less popular or for which the requests are more geographically spread out, can be mirrored at further away from the end-user, e.g. via Tier 2 peered locations [16]. Such a location can also typically serve a larger number of end-users and therefor can have a different statistical curve on which is decided what content is best to be cached.

The cache mechanisms used by the CDNs are only having any effect if they can generate a cache-hit when handling an end-user's request. Such a cache-hit is achieved if the cache already has the requested content object available because of an earlier retrieval.
This is only possible if content is requested multiple times through the same host that operates the cache. It is also important that these multiple requests for the same content come with high enough frequently, so that the content will be still in the cache the next time it is requested, instead being pushed out again already by other (popular) content.

When assessing the mechanism on which CDNs rely to optimize the distribution of content it is apparent that they can only improve content delivery to end-users if:

- A cacheable content object that is served to end-users is exactly equal for each recurring request
- a content object is served multiple times within the same network area,
- the requests for the content object are in high enough frequency within a limited time-window to survive in cache,
- the grand total of bandwidth usage can be reduced, which is the product of a file's size and how often it is served to end-users.

3.2 Virtual Machines

Virtual machines recently revived as a useful and popular concept when Stanford's former professor M. Rosenblum founded the VMWare company [17]. It is used to disconnect the physical aspects of a computing unit from the functional aspects of the computing unit by creating a new abstractive layer between the real physical computer hardware and the logical hardware that executes the computing instructions [18]. It allows to create a virtualized computing environment on a physical environment. To create this virtual environment a piece of software is necessary that is able to emulate the virtual hardware: the hypervisor, sometimes also called the virtual machine monitor (VMM).

There are different hypervisor implementations available from different vendors and each of those hypervisors has taken several different choices concerning some technical aspects during implementation [19]–[21]. Each of those differing aspects can make it even harder to find common structures in virtual machines as content; this is something to keep in mind when researching this subject. Because in effect there is no single 'virtual machine' content-type. Virtual machines should be considered more to be a concept, that encapsulates many possible implementations and variations, many of which will not even be discussed in this research. That's why sometimes assumptions are necessary or relatively arbitrary choices concerning the used technology have to be made when going into more detail.

In abstract terms, the following components on the physical host machine, on which the virtual machine is executed, are necessary:

- Hypervisor software
- Virtual machine disk image(s)
- Virtual machine configuration

Using these three components a virtual machine can be executed. The hypervisor reads the configuration file, creates a virtual hardware environment that is described in this file. The configuration file also points to the disk image, and this disk image file is represented as a hard disk drive (a block device) within the virtual hardware environment [22]. Within this virtual environment the virtual hardware is turned on and the contents on the hard disk are executed as if they would be on a regular (physical) computer.

Of these three components the hypervisor software is 'static', meaning that it only has to be installed once on the host machine and can be re-used for the executing of multiple (different) virtual machines. Meanwhile the virtual machine disk images and configuration are unique for each virtual machine. The configuration is just a few lines text and therefore not so hard to distribute. But the virtual machine disk image has a size, just like real physical disk, in the order of gigabytes. The transfer of such a disk image therefor poses the hardest challenge in the distribution of virtual machines between different locations and is therefore the logical focus of our research.

The hardest thing about virtual machine disk images, is that they tend to diverge and differ from each other: there is similarity but also a very high variance [23]. It could be

said that they are unique if considered to be per machine at a given moment in time. The cause of this fact is result of the following techniques that are used:

- Virtual machines use block devices for data storage
- Virtual machines tend to write continuously to their storage when executed
- Virtual block devices can be mapped in non-contiguous ways into a file

The first aspect, that virtual machines use block devices for data storage, is the first cause for the uniqueness of an individual virtual machine disk image. When a virtual machine stores information, it stores files on a filesystem. The filesystem handles the storage on the block device, but the filesystem is free to select in which blocks it puts which files, and files don't even have to be contiguously stored over blocks and can be spread out. This is also shown in Figure 2, where the yellow blocks make-up files, and the green blocks are the virtual block device:
The second aspect, that virtual machines tend to write to their storage, also creates the effect that the content of the files do change and therefor the content of the green blocks on the virtual block device. This makes the virtual block-device unique throughout time. Each time the virtual machine is executed, the contents of the virtual block-device change.
The third aspect, that virtual block devices can be mapped in non-contiguous ways into a file is influenced by two factors: time and hypervisor technology. Each hypervisor can choose a different method for mapping their emulated block device to a file or even a set of files [24]. This enables the possibility that different virtual blocks have a different location within the virtual disk image file. This is also influenced by time and the machine's history, the order of the data blocks can be randomly influenced by whatever location's blocks are written first to
This non-contiguous mapping is also visible in Figure 2 as is supplied by the VMware blog with the relations between the green and red layer, where the green blocks are representing the virtual block device and mapped to different locations on the red disk-image file [25].

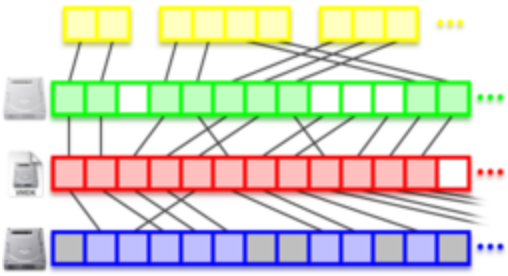

Figure 2 is an image from the VMware blog. It shows the mapping of disk clusters from a physical disk (blue), to a virtual disk image file (red), to a virtual disk (green), to a virtual filesystem (yellow). [25]

Focus on speeding up the process of retrieving a virtual machine disk image has often been on pre-fetching or on-demand fetching of the necessary data for execution [26]. But these solutions don't reduce the amount of total bandwidth consumption for the transfer of a whole virtual machine, or the distribution of multiple virtual machines.

3.3 Analysis of background overlap

From the preceding technical background about virtual machine disk images, it is apparent that there are many possible permutations for data blocks to be spread out over the disk image file. Also throughout time, when using the virtual machine, the disk image file can change.

This makes any virtual machine disk image file a unique object in bitwise perspective. This hinders the usage of current content distribution optimization techniques. Because the virtual disk image is by no means for them a trivially cacheable object. For them all virtual disk images appear to be different, they cannot effectively find similarities between various virtual machines, rendering their current caching and deduplication methods useless.

It also answers partly the first to sub-research questions. If a virtual machine cannot be cached, or deduplicated during distribution, it has to be copied directly from the source to the target location. That would mean the bandwidth consumption would be a multiplication of the network-distance between source and target with the amount of bits that have to be transferred for the virtual disk image file. The amount of bits would equalize the whole size of the disk image file, if no other revision of the exact same virtual machine is locally available.

4 Scenarios

In this chapter we depict three different scenarios of possible virtual machine usage that would desire distribution from one source to multiple target locations. Each scenario varies in context: what kind of copy is to be desired? Which content(types) are to be expected within the virtual machine? Which distribution factors are considered to be of importance?
The (details of the) scenarios and the list of performance factors per scenario are mostly result of a creative, iterative question-driven process. The first scenario is based on supervisor Strijker's work on Executable Scientific Publications and from this scenario a first set of performance factors were deducted [27]. Using these performance factors, other various scenarios with more varying results in these factors were sought after. The WebRTC relay server scenario is e.g. based on real world issues found by a TNO networking research group concerning WebRTC relay capacity. From the resulting set of scenarios other new impacting performance factors were introduced.

The scenarios and performance factors will be used to analysis the performance and applicability of the VMDN in chapter 6.6 in a variety of context.

4.1 Introduction of the Performance Factors

First of all, it is important to notice that even though the performance factors are here introduced before the scenarios, they were not fully established before the scenarios and VMDN prototype implementation were.
The performance factor originate partly from current virtual machine performance measurements in literature and whitepapers. Such as the time it takes from starting the download of a virtual machine to being able to execute it locally at a target location. Other performance factors were found after reviewing possible functionality impacting aspects for the usage of virtual machines in the VMDN's different strategies and the different scenarios and from the inner virtual machine image disk structure. Of course the set of scenarios and strategies were limited in number, which also implies that the list of performance factors may not be definite.

The performance factor list is (only for a overview) divided into two different categories: Virtual Machine Characteristics and Distribution Factors. The Virtual Machine Characteristics are performance factors that tend to be influenced depending on which kind of virtual machine and its contents is to be distributed. The Distribution Factors are performance factors that are more judged on given constraints within the applicable scenarios.

The performance factor's importance will be decided per scenario. The lowest score is 'Low' which means the given factor might is not considered essential or applicable to the scenario. 'Medium' is given to those factors that are considered nice to have, because of having possible impact, but not deemed to be essential. 'High' is given to those factors that will have a serious impact in the given scenario and could greatly influence the results of a successfully functioning solution for the given scenario. 'Critical' is given to the performance factors that, if not fulfilled satisfactory, would block the original design function in the given scenario. The importance score given to each factor is by reviewing the context, but as stated before, an endless amount of scenarios could be created with their own set of valued aspects and importance.

Virtual Machine Characteristics

- *Support and scope of VM variations*

  This performance factor describes the general support for variation in virtual machine image contents. Since virtual machines can contain a multitude of operating systems, filesystems and data contents. One of the strong characteristics of virtual machine technology is that by virtualizing hardware components, it is in general agnostic of the virtual machine's exact contents. But if you want to optimize performance by using tweaks, like hypervisors already do, this agnostic approach does not suffice. Hypervisors do use e.g. special disk, display and network devices that differ from the regular physical components that they would normally simulate [28]–[30]. By using special drivers in the operating system that deviate from regular hardware drivers these special tweaked virtual components can be used. E.g. if the virtual machine has an operating system for which these special drivers are not available, it won't be able to function at all. This limits the scope and support of virtual machine variations. The same restrictions might also apply to the VMDN, which could restrict the type of VM

variations that could be handled by the distribution system. This performance factor describes if such limitations would be likely to impact the given scenario or not.

- *Deduplication of the Operating System*

This performance factor describes the impact that deduplication methods would have in the given scenario for content that is part of the operating system of the virtual machine. This content type would be e.g. in Unix-like systems mostly the contents of /bin, /lib and /usr and on Windows systems C:\Windows\.

- *Deduplication of the Applications*

This performance factor describes the impact that deduplication methods would have in the given scenario for content that makes up the applications of a virtual machine. This content type would be e.g. in Unix-like systems in /usr/lib and /usr/bin and on Windows systems c:\Program Files\.

- *Deduplication of the 'User Data'*

This performance factor describes the impact that deduplication methods would have in the given scenario for content that makes up the configuration and data that is neither part of the operating system neither any of the static or data that is part of any application. In Unix-like systems this would at least include directories like /etc/, /home/, /root/ and /var/lib/. On Windows systems this would be at least c:\Users\.

- *Deduplication of Revisions*

This performance factor describes the impact that revision-tracking and -reusing methods would have in the given scenario. E.g. if a virtual machine would have each day a new revision to be distributed, but between every revision only one single file on the virtual machine's filesystem would be changed could greatly benefit from a tracking and reusing method.

Distribution Factors

- *Speed of importing VM image*

This performance factor describes the time necessary to import a virtual machine disk image file into the distribution network. This encompasses all the steps necessary, such as reading the original disk image, writing data, computational processing and any network activity.

- *Speed of reassembling VM image*

This performance factor describes the time necessary to retrieve a virtual machine disk image file from the distribution network and prepare it to be used by a hypervisor. This encompasses all the steps necessary, so reading data, any network activity, writing data and any computational processing.

- *Minimal size for effective cache*

This performance factor describes the object-frequency curve to indicate the possible effectiveness of a cache during distribution. The minimal size of cache can only be small only if a small set of objects is retrieved many times and thus can be served by the caching mechanism. If the distribution of object retrieval is more evenly spread out over a large set of objects, the minimal size for an effective cache would have to be much larger to be of any use. Also the size of the individual objects in the cache can (indirectly) influence the minimal size of the cache.

- *Re-usability of data-objects*

This performance factor describes the re-usability of the data-objects when reassembling a virtual machine image. If the re-usability is high, it would result in a

higher chance of already having a data-object available in a (local) cache. Thus it can improve the curve and hit-rate of these caches.

4.2 Scenario 1: Academic Research Publication

Our first scenario is about a virtual machine containing the tools and dataset use for academic research and accompanies a scientific paper and is based on the work of supervisor Strijker [27]. The idea behind this scenario is that in the academic world is important to be able to reproduce the research of an author. This to falsify or confirm any findings and to be able to perform any follow-up on the original research. Also it can be practical to be able to access any raw data that was used in a study from the past, for checking new insights.
Currently this reproduction can be hard, because of (details of) the original applications and datasets getting lost. Or because of incompatibility with newer hardware and software. The virtual machine could be a great solution to these problems. If all the necessary elements to reproduce the research are to be stored within a virtual machine, a study can always be retrieved later by just booting this virtual machine. All the applications and data available and directly ready to be used.

In this scenario the expectation is that virtual machines for academic papers will be in the future widely distributed as currently papers in the Portable Document Format (PDF) are. They will be offered as a download via the academic press to interested parties.

In the distribution scenario of these academic virtual machines there will be three stakeholders in the distribution process. The author of the research that created the virtual machine, probably a scientist. The publisher, probably an academic press organization that will publish the virtual machine. As last the consumer, probably a fellow scientist.

During the distribution process the following context is applicable:

- The author has plenty of time available for generation of the virtual machine
- The publisher has to host many scientific articles and their virtual machines.
- The retrieval rate of a single virtual machine will probably be relatively low.
- A peak in retrieval can be expected around the time of publication and presentation.
- The consumer will study for a longer time on the data anyhow and probably has plenty of time for retrieving the virtual machine.
- The consumer might have multiple alike virtual machines of related studies.

If we put this in an overview for which performance factors are important on a scale from low, medium, high to critical:

Performance factor	**Performance Importance**
Virtual Machine Characteristics	
Support and scope of VM variations	Medium
Deduplication of the Operating System	High
Deduplication of the Applications	High
Deduplication of the ‘User Data’	Low
Deduplication of Revisions	High
Distribution Factors	
Speed of importing VM image	Low

Speed of reassembling VM image	Low
Minimal size for effective cache	High
Re-usability of data-objects	Medium

Support and scope of virtual machine variations
It is to be expected that the virtual machines used for research purposes will reflect common variations as most other virtual machines. The popular hypervisors, operating systems and filesystems will dominate, exceptional choices will be rare.

Deduplication of the Operating System
The publisher hosting the virtual machines will probably have store hundreds of thousands of virtual machines, since 1.5 million new academic papers are published every year [31]. Since all of those virtual machines have an operating system deduplication for storage is essential to the publisher. We can note that the operating systems' versions might be diverse throughout all the virtual machines, because old studies will have older versions installed.

Deduplication of the Applications
The same applies to applications. When storing hundreds of thousands of virtual machines deduplicating the applications is important to reduce the amount of storage necessary.

Deduplication of the 'User Data'
Since the expectation is that between different studies the 'User Data' will always vary, deduplication is hard and probably in majority of the cases not possible and thus doesn't need any priority.

Deduplication of Revision Information
Because a study might be revised or spawn multiple articles for different publications, it might be useful to deduplicate on basis of revision information. Also for follow-up studies this might be useful.

Speed of importing VM image
The process of authoring an academic paper and the reviewing process is a time-intensive task anyhow, publishing the virtual machine for distribution is not time-critical.

Speed of reassembling VM image
Academic work is retrieved for study. Especially if the data is of interest to the consumer of the virtual machine it means they are processing the details. This is time-intensive anyhow and thus fast retrieval is not essential.

Minimal size for effective cache
Many different studies are retrieved each day from the academic press. There is meanwhile little chance that even a 'popular' paper with their associated virtual machine would be requested multiple times within a limited time period. It is because of this important that any caching mechanisms would be allowed to cache relatively small sets

of data compared to the complete set of all virtual machines for academic papers to be hosted. Otherwise the demanded storage for the cache would have to be high.

Re-usability of data-objects
Since the consumers of the virtual machines are probably working with a multiple of studies as their source of reference it might be advantageous to be able to re-use the downloaded parts, but it is not a necessity.

4.3 Scenario 2: Specialized Database Server

The second scenario is about a virtual machine that is dedicated for database storage. The context is that this virtual machine is distributed to various parties for a specific set of high performance database operations that it will execute. It contains an optimized installation of an operating system that is heavily stripped of unnecessary parts as possible, to keep as many resources as possible available for the database application. The filesystem is tuned to best performance for the database application. The data within the database is also carefully ordered on the filesystem and disk for best performance. The database application itself is compiled to be as fast as possible in the given environment. The results from the database operations are uploaded to an external database once a while. The virtual machine is replaced during scheduled maintenance windows with a newly distributed copy.

If we put this in an overview for which performance factors are important on a scale from low, medium, high to critical:

Performance factor	**Performance Importance**
Virtual Machine Characteristics	
Support and scope of VM variations	High
Deduplication of the Operating System	High
Deduplication of the Applications	High
Deduplication of the 'User Data'	Low
Deduplication of Revisions	Low
Distribution Factors	
Speed of importing VM image	Low
Speed of reassembling VM image	Medium
Minimal size for effective cache	Low
Re-usability of data-objects	Low

Support and scope of virtual machine variations
Since the virtual machines will have tailored filesystems and operating system it is important that these customizations won't block any optimizations for distribution.

Deduplication of the Operating System
Since the virtual machines are distributed on a regular basis with a new distributed copy and these copies mostly consist of the operating system, it is important that deduplication can be applied on this part.

Deduplication of the Applications
The same applies to applications. The application(s) used to run the database will make up the majority of the virtual machine that is distributed.

Deduplication of the 'User Data'
All 'user data' that is distributed with the new virtual machines is at most some structure information for the database application. This will be really small and negligible.

Deduplication of Revision Information
The content of the virtual machine instances to be distributed is heavily optimized for performance. This implies that there can be large (binary) differences between different revisions of files and applications. This makes revision information less useful. Also the virtual machine that is distributed is every time a new instance and doesn't inherit any data from the previous instance.

Speed of importing VM image
The process of creating the virtual machine image has to fit in the time between the scheduled maintenance intervals, assuming that on the intervals a new revision has to be ready and not an older one can be re-used. This will probably be plenty of time and thus of low importance.

Speed of reassembling VM image
The retrieval of the virtual machine image has to fit within the planned maintenance windows. There is a window, but it will have limited span. Retrieval speed is thus of medium importance.

Minimal size for effective cache
Since many same instances of the virtual machines will be distributed at planned intervals the retrieval rate will have a universal peak. The parties downloading the virtual machine will know about the distribution process and can install local caching mechanisms.

Re-usability of data-objects
As with the revision information, the virtual machines will be heavily optimized and probably have no or severely limited interchangeability of parts between different revisions.

4.4 Scenario 3: WebRTC TURN Server

The third scenario is about virtual machines that would be used to facilitate WebRTC connections between parties [32]. When parties cannot directly connect with each other because of Network Address Translation (NAT) an intermediate party is necessary to set-up the connection and (in worst case situations) relay data between the parties. Such intermediate relay is at best located on a network location that is close to all parties to avoid latency issues.
Since WebRTC TURN servers need to be configured per website and the parties needing the connection can come from all over the world, with variable demand. A great solution would be to run virtual machines providing the relay functionality at on-demand basis. Creating the virtual machine there where there is demand, and only if there is any demand.
These virtual machines could be created from a regular installed virtual machine, with a popular operating system using a normal precompiled package from the package management software for the WebRTC TURN server's relay functionality.

If we put this in an overview for which performance factors are important on a scale from low, medium, high to critical:

Performance factor	**Performance Importance**
Virtual Machine Characteristics	
Support and scope of VM variations	Low
Deduplication of the Operating System	High
Deduplication of the Applications	High
Deduplication of the 'User Data'	Low
Deduplication of Revisions	Low
Distribution Factors	
Speed of importing VM image	Low
Speed of reassembling VM image	Critical
Minimal size for effective cache	High
Re-usability of data-objects	High

Support and scope of virtual machine variations
Since the virtual machines will use common operating systems this factor is not of importance.

Deduplication of the Operating System
Since the virtual machines are distributed for a single functionality, it is to be expected that the operating system will be a large part of the virtual machine's content and should preferably be deduplicated.

Deduplication of the Applications
The same applies to applications. The application(s) used to run the WebRTC relay will make up the majority of the virtual machine that is distributed.

Deduplication of the 'User Data'

The ‘user data’ that is distributed will mostly be some configuration files, thus small and negligible.

Deduplication of Revision Information
These virtual machines are created to serve just for a short time (the period of demand) and afterwards be removed again. There is no expectation that any new revision of a virtual machine will be constructed.

Speed of importing VM image
The process of initially creating the virtual machine image only has to be executed when creating the WebRTC functionality on the website. This process is only once and speed is not important.

Speed of reassembling VM image
The retrieval of the virtual machine image has to be fast enough to meet the demand from connecting parties. This is probably the most important factor.

Minimal size for effective cache
Since the virtual machines will probably be retrieved at irregular interval over various places in the world, generating cache hits with a small cache is improbable.

Re-usability of data-objects
Multiple virtual machines for the same purpose are expected to be around. Plus the virtual machines are built from standard components. Being able to re-use existing data-objects should be useful for providing the desired functionality as fast as possible at a location.

5 State of the Art

This chapter is divided into three parts: first the related work to the structure of virtual machine images is discussed. On basis of this related work a brief analysis of the found virtual machine image structure is performed. Using this analysis of the structure, several structural layers and corresponding data-type objects are identified. For each of these data-type objects deduplication and efficient distribution methods are discussed in the third part.

5.1 Related work to Virtual Machine Image structure

Virtual machine image structure is apparently not a popular academic topic, since no literature directly describing this topic could be found. Partly this is probably because the most recent developments are more driven by industry than by academia. Another reason is that the image structure is mostly a result of (legacy) implementation of the storage stack. As a result this paragraph also relies on white papers and descriptions of current implementations.

5.1.1 Clusters

As provided earlier in the background chapter 3.2, the VMware company's blog had provided a nice diagram to showcase how data is stored using their VMDK virtual machine image disk format in Figure 2. Within this figure the storage of content within the virtual machine's image is displayed using blocks of data and lines for mappings between the layers. The location of the data differs per layer in the diagram. But each block and its (data) contents are always the same, as long as you follow the line of the mapping, no matter in which layer it resides. This is because in all layers this block represents a 'Cluster'. Clusters are a universal unit of storage. They are offered as a unit by most storage media, and filesystems use these clusters to organize their contents. Emulating the properties legacy storage media was chosen by hypervisors as method of providing virtual disks to virtual machines [26]. As a result, all advantages and disadvantages are automatically inherited.
The largest advantage is that the whole current legacy storage stack can be used. No special drivers or filesystems are necessary.
But this also introduces disadvantages. Virtual disks are not physical disks. But block device drivers and filesystems were designed with physical disks in mind [33]. This results in the effect that those designs that were optimal for physical disks, can actually be inefficient for virtual disks and negatively impact transfer speeds and access times [34].
But it can also pose issues when handling empty or by the filesystem unused clusters. Because when emulating a disk, also all parts of the disk that may not be actually in use (anymore) still have to be modelled by the virtual disk [35].

In conclusion: disk clusters are the basic element of storage within virtual machine images and they are used for storage on the level of the virtual disk image file, the virtual disk and the filesystems used within the virtual machine.

5.1.2 Files and Functionality

Within the diagram of VMware in Figure 2 the top layer is grouped into several larger sets of blocks. These sets of blocks represent files on a filesystem. It is the filesystem's responsibility to keep track of the clusters on the (virtual) disk to map, group and represent them as files to the operating system of the (virtual) machine.

The purpose of these files on the guest machine is in the end to perform certain functionality. Since virtual machines are objects that have to be created, it is fairly safe to say that they have a certain purpose. This functionality can only be performed by multiple files interacting together. Some files are part of the operating system or applications, while others are what we call here 'User Data'; data that is relatively unique per virtual machine e.g. describes its configuration, its state and could contain some files to be hosted by its webserver, or a relational database or anything else.

Based on these findings two types of storage elements were found: files, which are data objects managed by the filesystem; and functionality: which is the purpose for which the virtual machine was deployed and is enabled by using the files in an appropriate manner.

5.2 Analysis of Virtual Machine Image structure

The lowest structure of the virtual machine is made up by several layers that were already identified in the diagram of Figure 2 in chapter 3.2. All these layers were made up by data clusters as unit of storage: a data unit. Using the extra information found about the virtual machine image's structure this diagram can be expanded, which has been done in Figure 3.

This extension of the diagram is necessary for our research to look for possible distribution optimization methods. Since the old diagram did only have the data cluster as data unit. Thus optimizations could only be applied on these data clusters. By identifying more new data units, more possible optimizations could arise.

In Figure 3 two new data units are introduced along the data clusters. The files on the virtual machine and the functionality of the virtual machine are built on top of the stack of data clusters. A single cluster or a set of data clusters within the filesystem layer encompass a file. And one or multiple files represent functionality.

Only using this information about the virtual machine image's structure, the search for related work could be performed. The search focused on the three kind of type of data units that were found: Clusters, Files and Functionality.
The related work is discussed per structural layer where it could be applied

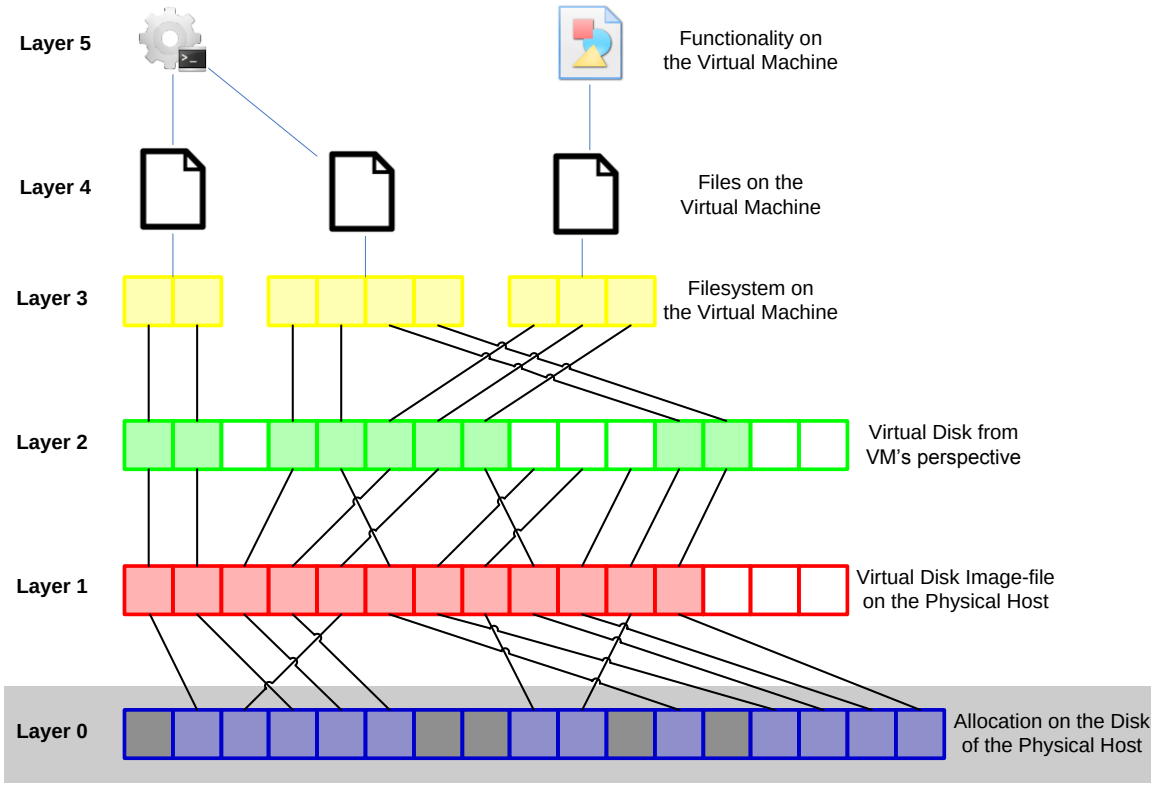

Figure 3 is a derived and further extended version of Figure 2 that shows the mapping of disk clusters from a physical disk, to a virtual disk image file, to a virtual disk, to a virtual filesystem, to files, to functionality. They layers are numbered from bottom to top. The bottom layer (zero) is grayed-out, since that layer will not be relevant to our further analysis.

5.3 Related Work to Virtual Machine Image distribution

The related work to our research are studies in a widespread set of research areas. Because this study touches multiple research fields that all play a role in the structure and distribution of virtual machine images: filesystems, file transfers, disk storage, deduplication, infrastructure-as-code and content distribution techniques.

The structure of the virtual machine is made up by several layers that were identified. The related work is discussed in order from the bottom to the top layer. For each layer we will try to identify the related work that is relevant to our research. First the related work to a file-based representation and distribution of the virtual disk image will be discussed. Then we will look into the related work to the cluster-based structure of the virtual disk. As third the relevant work for filesystem structure will be researched. As fourth we will look again into the file structure and as fifth into the functional properties.

5.3.1 Copying Virtual Disk Image Files

If considering the 'outside' virtual machine disk image no knowledge of the inner working of the image is assumed. The data that is being handled is a file that consists of bits that are in a certain order. To move a virtual machine disk image around in such a situation all bits have to be moved in exact order from a source to a target location.
For file transfers from a source to target location there are classical point-to-point transfer methods that copy bit for bit without any optimizations, such as Secure Copy (SCP) or File Transfer Protocol (FTP) [36], [37]. One of the bottlenecks for virtual disk image files transfers via such protocols is that sparse files (empty space within a file) are not supported and also the empty space has to be copied bit-by-bit making such transfers possibly inefficient [38].
Optimized transfers are possible using rsync, if the target location has an older revision of a virtual machine disk image [39]. Rsync compares with a sliding window the data at the source location with that at the target location. Differences are detected, compressed and transferred. Unfortunately also rsync also doesn't support sparse files if used with the compare-feature. Thus again all empty space within a file has to be transferred, though the compression filter is efficient with handling the long array zeroes. The largest disadvantage is that storage space will be wasted at the target location.
Virtsync is a fork (a deviation from the original rsync project by a new developer) off rsync [40]. This fork allows to combine the normally incompatible rsync command-line parameter *"--inplace"* for comparing both the source and target location with the *"--sparse"* parameter. Virtsync is therefore the currently most efficient method for copying a complete identical virtual machine disk image from a source to a target location.

5.3.2 Deduplicating Virtual Disk Image Files

For distribution scenarios data deduplication is often important. Within distribution scenarios, the default type of content object to be distributed are files. The most typical approach is to apply deduplication of content at the file-level. There are a couple of methods to apply on the deduplication strategies within files. The most basic approach is Whole File Detection (WFD), which hashes the whole file [41]. The hash is used for

searching duplicates in the storage. This is a simple and straightforward approach that is suitable for deduplication of static content. But more dynamic content, like webpages with randomized advertising, would have different hashes. To handle those cases specific cases for webpages Edge Side Includes (ESI) were developed [42]. But also more generic approaches like Fixed-sized partition (FSP) detection technology, content-defined chunking (CDC) detection technology and sliding block technology are available to deduplicate overlapping content over multiple files [43], [44]. But a problem with all of these technologies for deduplication is that it is necessary to download and store these content files completely before the deduplication mechanisms can actually be applied [41].
These traditional file-deduplication techniques can be applied on virtual disk image files, but as described in the problem statement and background, such deduplication is normally not effective on virtual machine disk image files.

Deduplication of is also used in shared-storage scenarios. Examples of such scenarios are the usage of Networked Attach Storage (NAS) solutions and distributed filesystems [45]. When using those technologies the filesystem on the storage device is shared via protocols at a file-level access manner. The client nodes can access these files as if they were on a local filesystem but don't see the underlying block-device storage [46]. The factual block-device handling is actually done within the storage node. This storage node can apply under-water deduplication techniques on a file-level or chunk-level basis. These storage methods are popular for two scenarios: a first scenario is with many different client nodes that rely on a common pool of files, to be accessed read-only. Examples are shared online disks with application executables or pre-populated data for users.
The second scenario is with multiple client nodes, which have to be able to read and write to a common dataset where each of the nodes has to be able to read what the other ones have committed.

Another approach to deduplicating content during distribution is to match hashes of chunks with local caches which can save up to 7% in overall transferred volume [47]. Data that is transferred between different hosts contains often redundant parts. When the data is transferred it is split up in different chunks, before sending the chunk from the source to the target, it first sends a hash of the chunk. If the target already has a chunk corresponding with the hash in its local cache, the chunk is retrieved from the local cache and not send by the source. Only if the target doesn't have the chunk yet, it is send.

5.3.3 Block deduplication

Block deduplication is also a possibility for deduplicating content. There are two basic scenarios for deduplication at the block level. One method is to leverage the current usage of clustered storage in Storage Area Networks (SANs). SANs differ from NAS that storage is made available not on a filesystem level, but at the block level [48]. In normal operation such SAN storage is inefficient for the deduplication of file-based content, since each client will claim their own disks for storage and no files or can be actively shared between the different clients and their operating system. But by not using any file-system level techniques but to detect just identical blocks within the storage cluster,

deduplication can be applied. The advantage is that this approach works agnostic of the filesystem(s) used, but the disadvantage is that a lot of metadata has to be stored and discarded blocks have to be garbage collected. But especially for the storage of (many alike) virtual machines that use the same storage cluster this can have positive impact in terms of performance (since identical blocks can be more easily buffered in memory) and in reducing the amount of storage needed [49], [50].
There is also an application from a research paper called Parallax that takes such clustered storage to a virtualized level [51]. It virtualizes the SAN nodes and puts the clustered storage within virtual machines. This allows the researchers to spawn storage clusters at the same physical hosts as where the virtual machines are run, which are the clients of the cluster. Also Parallax is able to deduplicate the stored blocks and stores the data in a so called shared blockstore. By using the state information available from the virtual machines, that are clients to the Parallax application, this information is already known to the host machine, thus it is able to housekeep the data more efficient than regular SANs.

The second deduplication scenario is by applying deduplication on virtual machine disk image files. With such an approach the disk blocks are not mapped directly from a physical disk to a virtual machine as in the storage cluster scenario, but the blocks are mapped to locations within a virtual disk image file. For such a scenario using classic virtual disk image drivers in the hypervisor have to be changed, or deduplication can only be performed in an offline manner. Research indicates that block deduplication for virtual machines sharing the same version and revision of the operating system and applications can allow 80% of deduplication of identical blocks [50].

5.3.4 Dirty-block tracking and overlays

Another method for deduplication is the use of dirty or changed block tracking (DBT or CBT) in combination with the usage of snapshots [26], [52], [53]. The basic idea is to only store the disk-blocks of any virtual machine copy, which have changed since it started to deviate from the original virtual machine.
Of course this method can only be applied on data that has a common ‘ancestry’. Which means that the started out with a single original version, which over time forked into multiple different revisions. This allows for an (relatively efficient) differences-only description of the virtual machine’s disk image.
The method is already widely used for virtual machine disk images, mostly because in practice there are two typical scenarios. The first scenario encompasses a data center where many virtual machines are created with a typical common set of functionality. The second scenario is with a single virtual machine that is mobile between different locations and is copied on a regular basis from one location to the other.
The first scenario is popular if the virtual machines that are deployed in a data center are pre-installed by a single party (like the hosting provider). In this manner they can prepopulate images with popular operating system distributions/revisions with popular (pre-installed) applications. This enables the delivery of such a virtual machine’s disk image as a snapshot. If many of the running virtual machines are based on a single

snapshot, all content on the snapshot can easily be deduplicated, as many disk-blocks are completely equal and can be shared by multiple virtual machines [54], [55].
Only the disk blocks that have changed on a virtual machine since the use of the snapshot that it originated from, have to be stored on a per virtual machine individual basis. This process is called dirty-block tracking. The blocks that have changed are stored in an overlay.
The second scenario often uses these overlays as a synchronization method. When copying a virtual machine from one location to the other, a snapshot is created. This snapshot is kept (in a static form) at the location where the virtual machine is being executed, but also at the other location(s). All changes to the virtual machines during its execution are stored in an overlay. When at a later moment the virtual machine has to be copied to another location, only the overlay has to be transferred, since the snapshot (which is already there) and the overlay provide all that is necessary to reconstruct the virtual machine's image.
This method is also used for the so-called "*live migration*" of virtual machines [53]. During the live-migration an initial snapshot is made of the virtual machine, all changes at the source location from then on are stored in an overlay. The snapshot is copied from the source to the target location. Once the copying was successful, a new (second) overlay-instance is created at the source and the first overlay is copied from the source to the target. Once this overlay has been copied, another overlay is made and the process is repeated. At some point, if there is enough bandwidth and the changes to the virtual machine are limited, the overlay becomes so small that it can be copied almost instantly. At that point, the virtual machine at the source location is 'stopped' and the instance at the target location is 'resumed', generating the effect that the virtual machine has moved from one location to the other in a short instance.

5.3.5 Filesystem optimization

Virt-sparsify is a tool developed by Red Hat's libguestfs-project [56]. This tool is able to read the partition information on a virtual disk and detect the installed filesystem on each filesystem. Virt-sparsify supports many filesystems and if it understands a filesystem it can scan which disk blocks are currently in use by the filesystem. Using this information it can also detect which blocks have been discarded e.g. when a file in the filesystem was deleted. The discarded blocks can then be zeroed or trimmed from the virtual disk image file by virt-sparsify. Also virt-sparsify can detected swap partitions and remove the disk blocks used by the swap partition from the virtual disk image.

5.3.6 Filesystem deduplication

In IBM's Mirage project a convertor was implemented that could import existing virtual machines disk images into Mirage's storage library for later export [57], [58]. The library imports block-based storage, thus virtual disks. But instead of only storing the blocks, an analysis of the disk is performed. Mirage detects the inner structure of the disk and analyzes the contents. Using this analyzing mechanism, the actual contents of the virtual machines' disks can be cataloged. The catalog keeps track of the partitions, the filesystems and files. Mirage's library thus is file-aware and can identify identical files to deduplicate their content. But it also allows for advanced governance of the virtual machines. Because it enables the possibility to keep track of which virtual machine has

what file, keep track of file-modifications between different revisions, and allows offline manipulation of these files. The disks can be exported from Mirage's storage library as block-based images, reconstructed from the library content.

Microsoft also performed a great analysis on file deduplication on many desktop computers [59]. The results were that storage consumption could be lowered with 32% by using chunk-based deduplication, where chunks varying in size from 8 to 64 Kibibytes. Around 25% storage consumption reduction could be achieved if only WFD-based (Whole File Detection) deduplication was applied, which only works for completely identical files. While chunk-based deduplication also allows for deduplicating files that share part of their content (of at least the chunk size).

5.3.7 File copying and deduplication

Using virt-mount from the Red Hat's libguestfs-project it is also possible to inspect a virtual machine image disk, find the partitions and detect the filesystems and mount these filesystems on a host machine [60]. After mounting these filesystems the files within the virtual machine are accessible like any other file, thus allowing using any of the file copying and deduplication strategies that were discussed before with the virtual disk images.

5.3.8 Infrastructure-as-Code

Infrastructure-as-Code is are relative new paradigm, it was coined around 2011 and has gained a lot of traction with popular projects as Chef, Puppet and CF Engine [61]–[64]. With the announcement of Amazon's EC2 in 2006 and the release of Ruby on Rails Luke Kanies announced the release of his research's project result "Puppet" [65]. The development of Puppet inspired developers to release Chef in 2009 really brought infrastructure to code. The philosophy behind this development was that infrastructure can and should be treated as code and that same professionalism and principles as used in software development could and should be applied to infrastructure development. Therefore, the many years of software development methodologies should be reapplied to the development of infrastructure as code.
Two important high-level steps are identified for developing infrastructure as code identified by Adam Jacob, creator of Chef:
1. Break the infrastructure down into independent, reusable, network-accessible services.
2. Integrate these services in such a way as to produce the functionality our infra-structure requires

These principles and new paradigm are now becoming key to virtual machine deployment. But unfortunately it doesn't completely embody an answer to our virtual machine distribution problem. But it might be a helpful part to a possible solution. The limitation of the current Code as Infrastructure's applicability is currently limited to the generation of the code. It needs manual human intervention that creates the code-recipes for the desired virtual machine deployment [66]. But it does have automatized the interpretation of recipes for creating the virtual machines itself. This while in our desired distribution mechanism the encoding of existing virtual machines into a recipe should be

an automatized process. Also Infrastructure-as-Code is focused on creating bare, new virtual machines, not yet in existence but only within the programmer's thought. While in our presented scenarios it would involve virtual machines that are already in existence. These differences are illustrated in Figure 4.

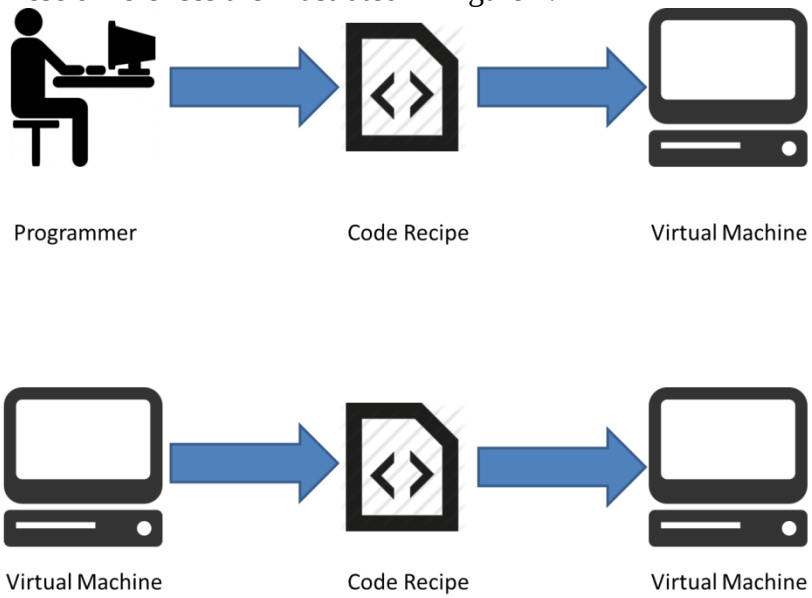

Figure 4 Currently Infrastructure-as-Code starts as a thought in the programmer's mind. The programmer writes the code recipe, the code can generate a virtual machine. In our scenarios the virtual machine already exists. For Infrastructure-as-Code for our scenarios the Code Recipe should be able to encapsulate a pre-existing virtual machine.

The arrows in the figure depict the processes of encoding and decoding (the concept of) a virtual machine into code and vice versa. Puppet and chef are providing the software that fulfills the right arrow, *'decoding'* the recipe into a virtual machine [67]. Also another solution is available that is able to perform the same transformation from a recipe to a virtual machine: Vagrant. Vagrant is developed with developers of (web)applications in mind [68]. Their goal is that you can create recipes for producing consistent virtual machines for development and production environments [69]–[72]. By creating a virtual machine from a recipe (the so-called *'provisioning'*) the idea is that it can be guaranteed that the environment in which the developed applications is being deployed is consistent, guaranteeing that when it works in a fresh provisioned development vagrant virtual machine, it will also work in a fresh provisioned production vagrant virtual machine.

The recipes for building the virtual machines for all three methods rely on bootstrapping virtual machine images that contain a base install. The base install is then booted by the hypervisor and a terminal connection like Secure Shell (SSH) is used to connect to the virtual host. Via this terminal scripts are executed in a scriptable Unix shell, like Bash. These scripts contain commands that edit configuration files and install packages via the package management system of the guest operating system.

But the *'encoding'* of a virtual machine is not available via puppet, chef or vagrant. Not even third party tools are available to do this encoding process. The only application that comes close to providing such a functionality is Docker [73]. Docker is much like

vagrant, chef and puppet a form of Infrastructure-as-Code. It downloads a base image to start with, it provides a consistent virtual environment for the execution of (web)applications and it can use recipes for reproduction of these virtual environments. But docker is not actually a virtual machine solution, as is illustrated in Figure 5.

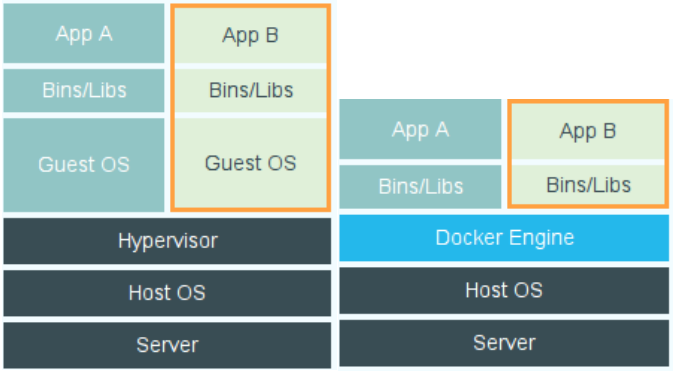

Figure 5 Copy of the diagrams from the Docker website, explaining "What is Docker" [73]. On the left is a visualization of the traditional virtual machine stack. At the right there is a visualization of how the Docker stack works and is indicated in yellow how Docker containers differs from regular virtual machines.

Docker doesn't use a hypervisor or any traditional (virtual) block devices as storage provider to the virtual machine, but instead it runs as a regular process and it gives the virtual environment access to a jailed chroot on the host's filesystem via its docker engine. This docker engine can record all changes to the filesystem that were performed during the execution of the virtual docker environment. By recording these changes, the docker engine can actually be used to create the recipe. The recipe then consists of a difference-descriptor for the files on the filesystem. By copying these descriptors, the environment created within docker can easily be replicated.
Docker also has the possibility of using DockerFiles. This are recipes as written for vagrant and puppet to be executed for docker environments [74]. These DockerFiles are, just like many other recipe languages, written as shell-scripts. These shell-scripts are text files that can be interpreted and executed by command-line shell applications like Bash.

Apparently Bash and other command-line shell applications are an import part of any the current recipe implementations. They are a powerful tools, because they allow access to many applications that is available within the virtual environment. The most important one of those is probably the package manager. The package manager is an application that manages the state of all the applications available within a given (virtual) environment, often the operating system. The implementation of package management with smart resolving allowed especially various Linux distributions to make the base operating system lean and allowed it to put all the middleware and other extra components in an on-demand distribution model [75]. This allows for basic configurations for virtual machines and their base-images with minimal functionality. When extra functionality is requested, the installation of packages is done on an on-demand basis. This allows tools like virt-customize to manipulate virtual machines using the package management software [76]. When functionality is desired from a virtual machine, virt-customize executes a shell script within the virtual machine that runs the package manager. The package-manager requests the packages from the internet and installs them within the virtual machine.

If there are multiple virtual machines requesting packages from the internet, there are solutions available to avoid requesting the package multiple times. Virt-customize and other tools have integrated package caching on the host running the virtual machine. But also solutions like Stork are available to share installable packages and avoiding the unnecessary retrieval of packages from the internet [77].

6 Virtual Machine Delivery Network

The Virtual Machine Delivery Network is a new concept that we propose in this research. It doesn't exist yet, but we know what it should do: it should have the functionality and strengths of a Content Delivery Network and then apply those to Virtual Machines as a content-type to be distributed. Hence instead of the agonistic term 'Content' in CDN, the choice for 'Virtual Machine' in VMDN. By being specifically focused on virtual machines as content-type it might be able to mitigate the pitfalls of that current CDNs experience for the distribution of virtual machine images.

6.1 Virtual Machine Structure

For the VMDN it is key to leverage the understanding of the virtual machine's image structure into finding content objects that are better candidates to be cached than the original virtual machine image.
The structure of the virtual machine is made up by several layers that were already identified in chapter 5.2. At the lowest layer little information about the inner workings is necessary but optimizations are limited. But with each following layer more understanding of the used technologies is necessary to extract the cacheable objects. Meanwhile, for each of these layers different replication strategies can be used and different implementations are necessary to extract the structural objects that make up the virtual machine. Also a choice for a specific replication strategy implies that a copy of the virtual machine is equal on the associated layer of that strategy, but that lower structural layers might differ.

In this chapter we will first discuss the general choices for replication; a copy can be based as replication of the original virtual machine at certain level of the structural layers that were identified. Then the different replication methods of these layers, the associated strategies and the VMDN prototype implementation are discussed in three separate sub-chapters. These three chapters organized in line with Figure 3 from chapter 5.2. Within this structural scheme six different layers were identified, each representing a structural layer of a virtual machine. Layers 1 to 3 are all using disk-clusters as a data unit. These layers and a strategy that matches these layers and its unit for a possible VMDN strategy will be discussed in sub-chapter 6.3.
In the second layer from the top, layer 4, files are used as a data unit. For this layer we have a matching strategy we can be found in sub-chapter 6.4.
In the top layer, layer 5, the virtual machine's functionality is the data unit. For this layer we have another specific strategy which will be discussed in the last sub-chapter 6.5.
In each sub-chapter the technical background of the layer is analyzed. The opportunities for replication, the benefits and possible issues are discussed and a design for the VMDN strategy is presented. Using the design a prototype implementation is introduced. Also the pitfalls during development of the prototype will be mentioned and limitations that are a result of the general design.

6.2 Replication choices

For replicating the virtual machine image from the source location to the target location, it is important to note of which layer the copy at the target location should be a copy. The lowest layer in Figure 3 (layer 0) is not relevant for this study and can be ignored. Then from the bottom-up the next three layers can be identified (which are numbered 1 to 3) that all use data-clusters as a unit of data-storage:

1. The virtual machine image disk-file, also known as a "binary outer" representation of the virtual machine,
2. The virtual disk block device, also known as a "binary inner" representation of the virtual machine image's disk,
3. The virtual machine image's filesystem(s), which are the data-clusters that are actually in use by the filesystem.

On top of these three basic layers we find the two other layers that were identified:

4. Files, the files that are stored within the virtual machine,
5. Functionality, the packages and data or other forms of descriptions that describe and can summarize the purpose of the virtual machine.

When replicating a virtual machine at a certain layer, it may mean the underlying layers are not equal between the source and target. Depending for which purpose a virtual machine has to be replicated this difference might have an impact or not.
This is a question should be considered as part of the context as was depicted in chapter 2.1 with "*what is a copy?*" Per scenario the "function" of a virtual machine might be at a different structural layer within the virtual machine, depending on how much or if any of the lower-level technical implementations are considered of importance.

6.3 Strategy 1: Disk clusters

6.3.1 Technical Background

The first strategy is based on the technical design of the virtual machine philosophy. Because philosophy behind virtual machines is based on re-implementing physical hardware in a virtual setting it is possible to focus on the design principles of these hardware components. For storage the major hardware component used to be block devices, like hard disk drives. The unit of allocation of such a device is a block, or the more modern and universal term 'cluster'. A cluster is a set of bytes that together forms the smallest allocable amount of storage on a storage medium. Filesystems store their data per cluster and keeps track of what is stored in which cluster. Thus the cluster is a unit of storage.

Two great properties of this unit is that it is universal, all virtual and physical machines use it, and that is agnostic, it is just a container of a fixed amount of bytes and ignorant of their meaning. This makes this a collection of these units a practical one for storing any kind of (virtual) machine and any kind of operating system or filesystem. As long as this unit is used, any machine and its content can be handled.

Unfortunately this unit of storage also has its drawbacks. Since it is agnostic about its content, it is not known if the data that is stored within a cluster is still functionally in use or not. Also the unit is relatively small, for most (virtual) disks the cluster size is 64 KiB. This implies that a single disk of several gigabytes consists of millions of clusters.

It is known that the majority of currently popular filesystems (like FAT32, NTFS, ext4) organize their file data over the clusters in such a way that a file can only start at the beginning of a cluster, a file that is spread out over multiple clusters is always cut-off at the same fixed locations within the file (linearly with the cluster size) and at most one file is stored per cluster and any unused space at the end of a cluster consists of zeroes.

These properties of clusters and the inner workings of filesystems result in the effect that if two disks contain identical files and the same filesystem all content within the clusters on the disk that are in use (besides the where the File Allocation Table (FAT) is stored) are equal, only their order can be different.

This property can be used for deduplication of content. Disks with the same filesystem that share many equal files will also share many equal clusters. Thus popular files that are to be found in many virtual machines will also result in popular set of disk clusters. Also files that are recurring multiple times within one filesystem will result in multiple equal disk clusters.

These technical properties with disk clusters as a unit of fixed length, content agnostic and recurrent within and over multiple virtual machines' disk images should allow possibilities for optimized deduplication, caching and distribution.

6.3.2 Design

The design of a disk cluster based deduplication, caching and distribution mechanism is based on several steps:

1. Analyze a virtual disk image and extract properties like where in the image file the actual disk clusters start (the offset) and the disk cluster size used.
2. Split the image into many parts, each part containing a single cluster from the disk.

3. Hash all the content of the clusters. Store for each hash the location of its cluster on the disk.
4. Put all clusters' data and their hashes into a database. Put all locations of hashes of disk images into another database.
5. To extract a disk image from the storage, the hashes for all disk locations are collected from the database and the disk cluster with the corresponding hash is retrieved.

Using this mechanism a disk image can be imported for deduplicated storage and exported for deployment or distribution. The unit of storage is a cluster, so all stored objects are originally 64 KiB and have a hash, e.g. if a MD5 is used for hashing such a hash would be 128 bits [78], [79].

A virtual machine disk image distribution mechanism could also be cluster-based, using an alike approach. A target location requesting a virtual machine's disk image would first have to retrieve the database with all hashes that are used on the disk and their on-disk location. Such a database will be small in size compared to the original disk image and is unique per virtual machine. This database itself is not a cluster-based file, but a regular file and thus can be compressed using regular file-compression methods. Also regular file deduplication and caching techniques could be applied for distributing the database file to target locations. The target location will after receiving the database reconstruct the disk-image. These steps, from importing an image, to reconstructing it, are summarized in a flow-chart in Figure 6.

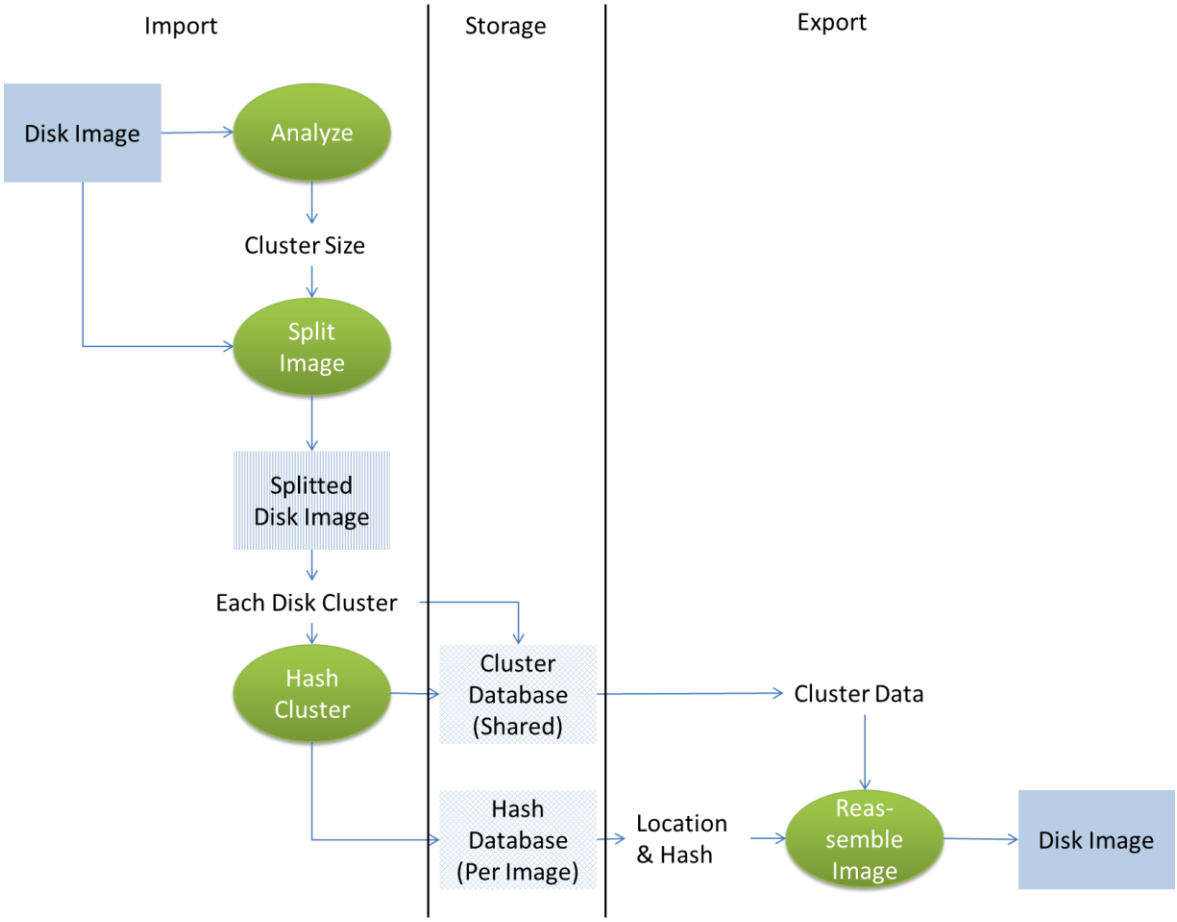

Figure 6 A flow-chart showing the steps that are followed for strategy 1. The disk-image is analyzed and split into many small clusters of data. Each cluster is hashed. Each hash is stored with its found location in a database that represents the image. A hash entry is created with the disk data in a central database containing all the clusters.

The reconstruction processes can be performed in two different manners: preloaded or on-demand. With the preloaded technique all hashes are read from the database for the whole disk image. The target location will have a local cache for hashes and disk data. It will try to find the data of the hashes it already has from its local cache. The disk data from the hashes that are not available in the local cache will have to be retrieved. For the retrieval of this data a CDN-like infrastructure could be used. Popular hashes that would be requested frequently could be cached on nodes near the target location(s). Less popular hashes could be retrieved from caching locations further away or even from the source location, where the virtual machine disk image originated from. Using the disk data collected using this mechanism the whole virtual machine's disk image can be reassembled at the target location. The disk image could be provided as a regular virtual disk image file to current hypervisor software, or a new intermediary virtual disk image driver could be written for the hypervisor. Allowing the hypervisor to request the disk data directly from the database. In the latter case it would allow for data deduplication for multiple virtual machines disk images on a single physical host.
If using such a new virtual disk image driver for the hypervisor, it would also allow for the on-demand disk data retrieval scenario. Since with the database the structure of the disk image itself (e.g. the size) are known a dummy disk image could be easily be created

for booting the virtual machine. During the execution of the virtual machine, the disk locations of all I/O transactions to the virtual disk image are picked up by the hypervisor's virtual disk driver. The disk driver fetches the corresponding hash for the location from its database and checks if it has this data already available. If not, the data is requested from a remote location and added to the database, just like in the earlier pre-fetch scenario. Writes that are made to the virtual disk drive have to be directly hashed and stored in both database. Using this method would allow to reduce the total amount of data that has to be received by the target location, if not all contents on the virtual disk drive are used. But obviously it can generate high latencies for disk I/O if disk data is not available in the local database. Possibly this problem could be partly mitigated by using smart pre-fetching algorithms that could (partially) predict ahead in time which disk locations might be requested and already fetch the data from these disk locations.

6.3.3 Implementation

For the prototype a simple implementation was built in Java. For the simple prototype only the pre-loaded scenario is implemented, using a regular local file storage. Physical transfer of the virtual disk image data over the network and usage of the described on-demand scenario are beyond the scope of this prototype. The implementation was focused on handling QCoW2 (QEMU Copy-on-Write 2) and raw disk image files [80]. The prototype can check if a file is a QCoW2 disk image file by checking the file's header. If the file is a QCoW2 file then the cluster size of the virtual disk is also read from the file header and used.
Because splitting the data really per cluster would create too many entries to efficiently handle in the prototype's data-storing database a more pragmatic approach was necessary. In the implementation the cluster size that will be used in the storage shall not be the size of a single cluster. Instead this will be a multiple of the clustersize, e.g. a multiplication of eight. This number was found to be relatively optimal in other studies, creating the best trade-off because of barely any decrease in finding hash matches [50]. The source of this relatively limited negative impact is because most writes to clusters are linear. The resulting effect is that a single file often spans multiple clusters, which are linearly written during the single file write-transaction. Combined with the fact that many disk-transactions are identical over multiple virtual machines, because they are performed by e.g. scripts and installers. Also most filesystems buffer their write-transactions before writing to the disk, to obtain better disk lay out. These three characteristics contribute greatly to the general result that not only single clusters are often to be found identical over different virtual disk images, but that even sequential series of clusters often are. Thus even though by using this property the amount of possible deduplications are reduced a little, it greatly enhances performance and also the efficiency, since the same small hash for identification can now correspond to even more stored data.

The QCoW2 file uses a mapping mechanism, since not all clusters are stored at the same file's inner physical location as they are stored on the virtual disk image's location. To track these mapping of the clusters, so-called mapping tables are necessary. These mapping tables also have to be stored within the virtual disk image file. These tables are spread out through the disk image file at various locations. An overview of this mechanism is given in Figure 7.

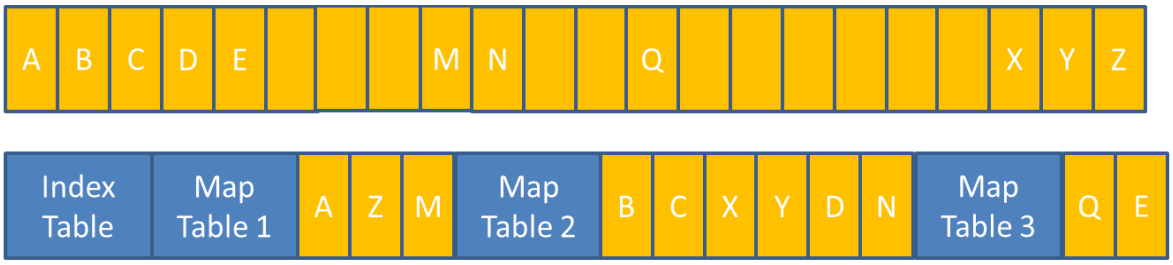

Figure 7 An example ordering of how data might be ordered within a QCoW2-file. At the top there is the representation of the virtual disk as it is visible to the guest virtual machine. At the bottom is the storage as could be found within the QCoW2 files. The mapping tables are stored between the regular data and the data itself can be stored in any order.

But luckily the QCoW2-format has been designed in such a way that these tables are the same size of a disk cluster. The prototype implementation therefore just ignores the existence of these tables all together and handles them as if they would be any other regular kind of disk cluster.

The Java prototype implementation depends on two external libraries:

- MapDB for database storage [81],
- DSI utilities for handling large files [82].

MapDB is used as database storage. This library offers an API like the regular Java Map-implementations, but instead of storing the Map only in memory it writes it to disk. This allows for a trivial database implementation where the Java Objects in the Map are written to disk (but only if these Objects are serializeable). In the prototype two database formats are used, a shared one for storing the disk data of the cluster in conjunction with an identifying hash. The other format is a database per virtual disk image, storing the locations of the clusters within the disk with the corresponding hash. Both databases are implemented using the HashMap equivalent in MapDB. The advantage for using this HashMap, that is technically a HashTree according to the MapDB documentation, is that it is fast for lookups if the key is known. Since the key is always known in both of our search scenarios; having a disk location or hash as search-key, performance is good when reading values from the database. But when adding new (not already existing) disk cluster data entries to the database it might get slower as the database grows.
The DSI utilities are used for reading and writing to the virtual machine disk image files. Since the disk image files are raw data, the prototype has to be able to read and write bytes to and from the disk image file. But java has set the limitation for its regular file bytestream I/O-operations that only allows memory addresses (locations within the file) to a maximum of Integer.MAX_VALUE which is a value of 2^31-1. This is done for compatibility reasons, but poses a problem for addressing the bytes beyond the 2 Gigabytes in a file. The DSI utilities solve this issue for the prototype by mapping multiple bytestreams in their backend on a single file and presenting them as a single bytestream through their API. This allows the prototype to work with virtual machine disk images that are larger than 2 Gigabytes.
To store the disk cluster data within the database an object was necessary that can contain bytes is serializeable. Since the ByteBuffer class in Java is not serializable a wrapper was written to put byte arrays in an object to store the disk data within the HashMap.

The prototype uses MD5 as hashing algorithm. This algorithm has been relatively arbitrarily chosen, but is well supported by all Java Virtual Machine implementations and the chance on hash-collisions is very low. Since the hashes are always 128 bits long and byte arrays cannot be used as keys in HashMap, a wrapper class was also necessary to store the hashes. When writing this class a direct easy performance optimization was possible. Instead of storing the hashes as byte arrays, like is normal, within the wrapper Object, it is stored as a combination of 2 long values. Since longs are 64 bit and the hashes are always 128 bit, this allowed for an easy but great performance optimization for search operations in the database Maps.

6.3.4 Usage of Prototype

The prototype can be executed by importing the binary-analyzer Java-project in Eclipse. All dependencies can be automatically resolved using Maven's dependencies management. Within the RebuildFromCache.java five static Strings can be defined. The DBfileName is the location of the file which we will be used to store both the hash and cluster database. This file is persistent and can be continuously reused when importing virtual machine images. The four other entries are parameters that should be changed according to which virtual machine images should be added to the cache, or which virtual machines are to be extracted and rebuild. If the application will be implemented beyond a prototype, such parameters should be settable using an interface.
When the prototype is executed to import a new disk-image it creates a Java-Object that encapsulates all the properties of the disk-image file. The properties are partly set by analysis of the file. E.g. it is known in the QCoW2-file format where within the file the cluster size of the virtual disk can be retrieved. For each different virtual disk image file-format that would have to be supported, such a corresponding Java-Class has to be created.
The encapsulated Java-Object for the disk-image is passed on as a parameter to CacheDB's addToCache-method. addToCache retrieves the properties from the passed disk-image object and creates a new Hash Database for this disk image. It then iterates over the disk-image file, hashing each (set of) cluster(s) and then adding the hash and the data from the cluster itself to the shared Cluster Database. The hash is then also stored with the location within the disk-file in the Hash Database.
After adding a disk-image to the database it can also be retrieved. This is done with the rebuildFromCache-method. For that method the original name of the disk-image, which is used as an identifier, and a target filepath have to be specified. Using the original name the set of properties and all locations and their hashes are retrieved from the Hash Database. Iterating over the retrieved set of hashes the corresponding cluster data is retrieved from the Cluster Database. This data is then written into the target file in a sequential manner.

6.3.5 Limitations

The prototype has several limitations. The first limitation is connected to using the multiplier for handling several clusters combined within one data-Object that is hashed. If using an 'unlucky' offset this can result in potentially a lot of mismatches for the data. This problem could be partly avoided by choosing offsets relatively dynamically, trying several different offsets when adding the disk image to the database, choosing an offset that would give the best result. Also, because of the multiplier, it is important that data

within the filesystem is relatively in-order on the clusters. If files are heavily fragmented within the filesystem it is harder to match them to the database. This might be partly resolved by defragmenting each filesystem before adding the virtual disk image to the database, increasing the chance of alike in-order layout over the disk's clusters.
Another issue is that the database with disk data would get slow for adding new data clusters with their hash when it would become large. Especially noticing that the data clusters are relatively small data sets, this could seriously hurt performance if a lot of disk data would have to be processed.
Also the prototype is at the moment only focused on local application of the strategy mechanisms, no network functionality has been implemented.

6.4 Strategy 2: Files

6.4.1 Technical background

The second strategy relies on comparing and caching files. In contrast to strategy 1, this strategy is not aware of any of disk clusters or virtual disk file formats whatsoever. That is because this strategy operates on a higher structure of the virtual machine's image. To be able to act on this higher structural layer within the disk image, the image will have to be analyzed and understood by the software handling this task. This poses extra challenges, but can also directly solve some of the inefficiencies compared to solutions that are focused on a lower structural layer.

The lower structural layers of the source disk image will have to be summarized in a meta-description so that the necessary conditions for the storage of the files can be recreated at the target location.

The lower structural layers that would have to be in the meta-description are in order from bottom to top of the reference Figure 3:

1. The virtual disk image file format
2. The virtual disk's capacity
3. The partitioning of the virtual disk and the contents of the master boot sector
4. The filesystem format and parameters of each partition

The lower layers can be recreated at the target location from this meta-description and offer a platform on which files can be stored.

A big advantage for this strategy is that many files that are found in various virtual machines tend to be exactly equal. Think about files for the operating system and the applications. But also data can be equal over vastly different virtual machine installations, like graphics or video files. While these equal files might be stored in different ways in the lower structure and therefore be hard to deduplicate in strategy 1, with this strategy instead they are easy to handle for deduplication. This can result in much more successful matches and improve performance.

Another advantage is that for regular file deduplication and caching already many methods are available. So there is no need to reinvent the wheel and those current methods can easily be applied.

A third advantage is that for file deduplication also changes-only file-description is possible, which allows to synchronize files that have multiple instances with just (small) differences between various revisions. These files could relatively easily be identified since of each file the full name and the path are known, which is useful meta-data for identification.

6.4.2 Design

This strategy relies on interpreting the virtual disk image file as a virtual disk, detecting the partitions on the disk and mounting the filesystems on these partitions. So first the virtual disk image file format has to be interpreted. Since there are many different virtual disk image file formats available, mostly depending on which virtualization hypervisor is used, this poses a challenge and a question.

First the challenge, which is being able to support as much virtual disk image file formats as possible when reading a virtual disk image file at the source location. The specifications of several formats can be found in public documentation, but for other

formats external tools may be necessary. The question is, whether you would want to recreate the same virtual disk image file format at a target location. This choice is probably mostly correlated with the question if you use the same hypervisor at the source location as at the target location. If this is demanded, support for a virtual disk image format would also have to be implemented in the software that recreates the disk image file at the target location.
After having a virtual disk available it is key to get the partition data and the boot sector from the disk. The boot sector should be copied as a file with binary data, the partition table could be in binary or in textual format.
Using the information from the partition table a bootable disk partition should be detectable. It is key that only one operating system is active and that there are not multiple bootable partitions. The filesystem on the bootable partition should be detected and mounted. Using the information found from the operating system and/or bootloader information at this first mountable partition the information about the other partitions can found and also be mounted at their corresponding path, or on purpose be ignored if they represent e.g. (outdated) swap data. Also this information has to be added to the meta-data.
After these steps a complete tree of filesystems can be mounted with all correct paths for all files that can be found on the virtual disk and its filesystems. From this tree a full list can be compiled of all files, their full path, security properties, their size and a hash. All of these files can be added to a regular file storage database for further distribution. The file list and the earlier collected meta-data are saved as a unique descriptor for this virtual machine.

At the target location the file list and the meta-data are downloaded. Using the meta-data a virtual disk image file can be recreated. This virtual disk image can be mounted and then partitioned with the partition information from the meta-data. The boot sector has to be copied from the meta-data to the virtual disk. The partitions are to be formatted with the file-systems described in the meta-data and mounted at their appropriate mount points. Then the file-list can be read and all files can be retrieved. For this retrieval current content delivery network techniques can be used and a local cache. Also, if a file with a matching path and filename are found within the local cache, but the size and hash do not match the file could still first be initialized from the local cache and then updated from a remote source with the correct revision using a difference-only file syncing method.
After the whole system at the target location has been brought up to date with all the files the security properties can be set on all files in the filesystem.
The result of this method is a copy of the files on the target location as they were on the source location. The inner-order of the technical representation on the filesystem can differ from the target and the source. But the outer-representation on the filesystems is exactly the same. A simplified version of these steps are depicted in

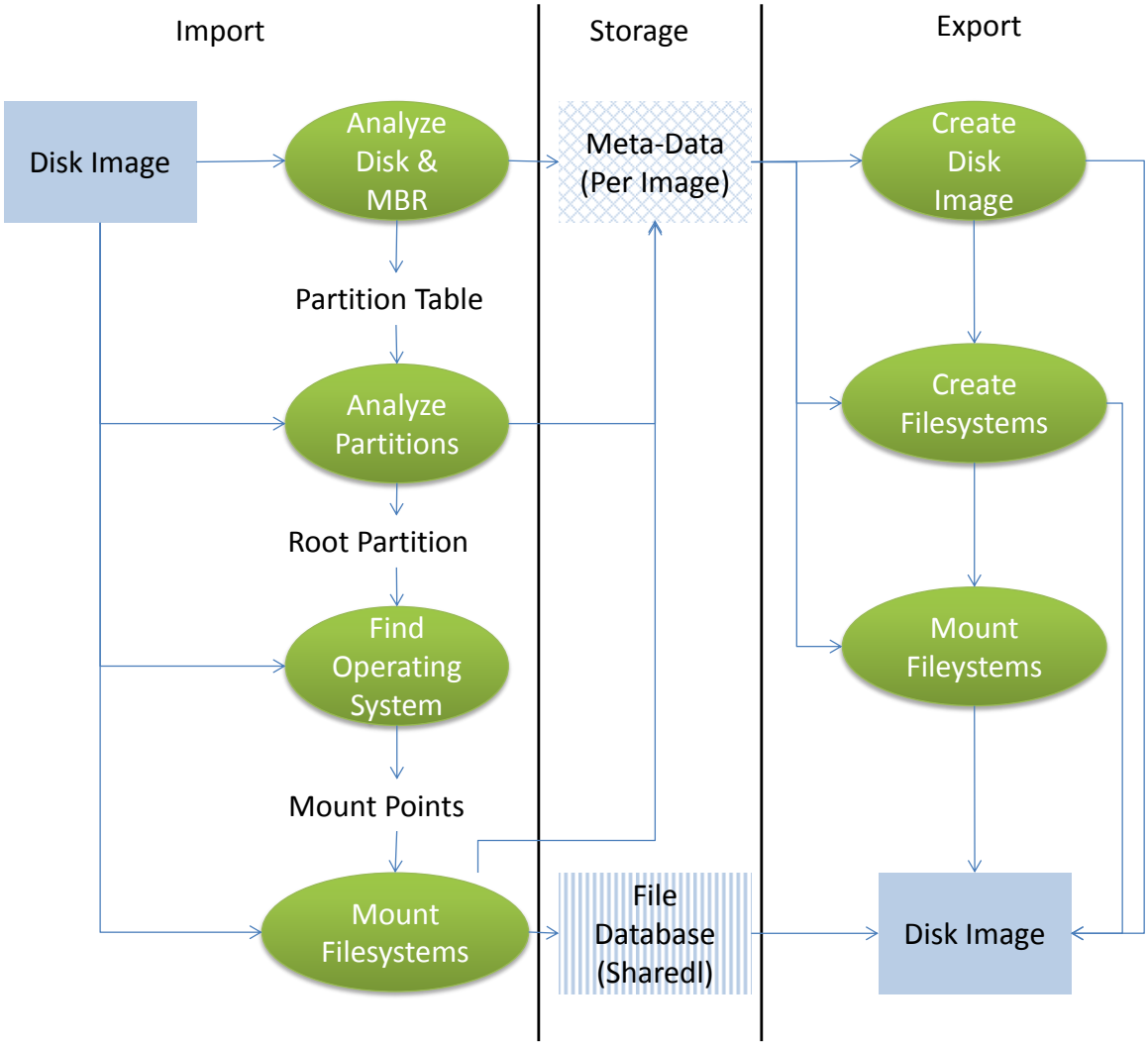

Figure 8 A flow-chart showing the steps that are followed for strategy 2. The disk-image is analyzed for its partition information and the MBR. The partitions are analyzed for their size and their filesystems and mountpoints are retrieved. These data are stored in the Meta-Data file. The filesystems are mounted at their respective mountpoints and the files are copied to the file-database. During reconstruction of the image, the disk image and filesystems are created. These are mounted at their respective mountpoints, the files are copied from the database onto the disk image.

6.4.3 Implementation

For the implementation of the prototype the usage of QEMU in combination with libguestfs, the FUSE mount system, the rsync synchronization tool and a bash shell environment were used. The target virtual machine images for the prototype were default installations of Ubuntu Linux 13.10 using a single ext4 filesystem and a separate swap partition.

Libguestfs provides a command-line tool, called *guestmount*. Guestmount uses virt-inspector of libguestfs to auto-assess a virtual disk image for its partition information, bootable partition and installed operating system. This information can be outputted to a file and be used to create the meta-information about the virtual machine's disk image. Guestmount then starts a virtual machine appliance using a simple Linux environment (called '*supermin*') within the KVM virtualization environment [83]. Within this virtual

machine it mounts all partitions from the virtual disk. It then exposes the whole filesystem tree via its API to a FUSE module that allows the host machine to see the filesystem of the virtual machine from the host's own perspective.
The prototype has a bash script that can run on the host machine. This bash script can copy all the virtual machine's filesystem data to a simple central file storage on the host machine. The prototype doesn't implement a central database mechanism for the file storage, but this could be trivially achieved by using one of the many drop-in solutions available.
The prototype also contains a bash script that uses rsync to make a file description of all the virtual machine's files. The output exists of two files: one large file that contains all data from the files on the filesystem in rsync's own (compressed) data-stream format. And a second file that contains a bash script that can, when executed, execute rsync with the correct set of parameters to reconstruct all file data from the large data-stream file. This rsync method also allows to be applied on multiple (but different) virtual machines where files are (expected to be) alike. If you have a virtual machine A and a virtual machine B, an output of the data-stream would only have to contain the differences between the files on A and B. If you'd then already have an (file-identical) instance of virtual machine A at the target location, it would be possible to create virtual machine B from the data-stream, or vice versa.

Using KVM virtualization for mounting and accessing the virtual disk in the prototype gives some overhead, but is much safer than directly mounting the virtual disk directly in the host itself. It would be comparable to plugging in an USB-stick to the host, exposing the system to possible malicious low-level attacks. By using the virtual environment as a sandbox this risk is mitigated.

At the target location a new virtual disk image can be created using the qemu-img from QEMU. Using the virsh tool from libguestfs new partitions can be created on the virtual disk. And using virt-make-fs from the libguestfs tools the filesystems can be created on the partitions. Again with libguestfs's guestmount in conjunction with FUSE the virtual machine's filesystems can be mounted and be accessible to the host. The host can put the files into the virtual machine using the file synchronization methods like rsync.

6.4.4 Usage of Prototype

For the prototype the execution of this strategy is currently done through a manual sequence of command-line operations.
The analysis of the virtual machine disk image is performed with:

```
virt-inspector --ro -a oldimage.raw
```

This command returns an XML file with disk, partition and filesystem information. These properties should be noted and stored in the meta-data.
Next step is to mount the old image through guestmount:

```
guestmount –r –a --pid-file guestmount.pid oldimage.raw
/tmp/mountedimage/
```

At this point all files of the virtual disk are available at the /tmp/mountedimage/ path. These files can be copied using cp or rsync to a remote storage:

```
cp –r –a –u –i /tmp/mountedimage/ /var/database/image1/
```

This would copy all files that exist on the source disk image into the target location.

Afterwards the virtual disk can be unmounted with:

```
guestunmount /tmp/mountedimage/
```

For creation of the target disk image, the following steps are necessary:
First the virtual machine image disk file has to be created. This should be done using the meta-data that was retrieved earlier, and could e.g. look like the following command:

```
qemu-img create -f raw -o preallocation=metadata /tmp/newimage.raw 2G
```

(NB: it is important to notify that the preallocation=metadata is not related to our own set of 'meta-data', it is just a qemu-img commandline option!)

Using the meta-data also the filesystems should be created on the disk, an example could be:

```
mkfs.ext4 –F /tmp/newimage.raw
```

When the disk, partitions and filesystems are recreated, it can be mounted. This would be trough the usage of guestmount like:

```
guestmount –a –i --pid-file guestmount.pid /tmp/newimage.raw
/tmp/mountedimage/
```

When the image is mounted, the file contents could again be copied from the database to the image, like:

```
cp –r –a /var/database/image1/ /tmp/mountedimage/
```

6.4.5 Limitations

The prototype was written with Linux hosts and Linux guests in mind. Though it should work without any problems with simple windows guests, the host machine does need Linux, because libguestfs is only available for this platform.
The kernel image file has to be readable by the user for guestmount to work. This demands a change of file permissions on Ubuntu systems (see libguestfs.org/guestfs-faq.1.html for more information). The user executing the guestmount applications needs access to the KVM kernel subsystem and FUSE kernel module, so it is necessary that this user be in the appropriate (c)groups of the host. Setting these permissions demand more administrative overhead and extra security management than regular executable applications.
This method can only be applied on virtual machine images of which the virtual disk format is supported by libguestfs and where the partition formats are understood by the supermin application and the file-properties and permissions are understood by the host's operating system.
It is important to note that within current file distribution and deduplication infrastructure there is in general no security and access management available per file. This might be of importance if files within the virtual machine contain intellectual property or private

information such as passwords. E.g. for these situations it is important not to use public accessible CDN infrastructure for file-distribution.
Also the prototype is at the moment only focused on local application of the strategy mechanisms, no network functionality has been implemented.

6.5 Strategy 3: Functionality

6.5.1 Technical background

A third strategy depends on the functionality a virtual machine encompasses. With the trend to make multiple lean virtual machines for one or a small set of tasks instead of a single virtual machine handling many, a virtual machine tends to be kept simple to perform a limited and generalized set of functionality. To offer this functionality often a single or a few applications are installed. But these applications still depend on an installed base system. This base system is an operating system and all kind of support libraries that applications depend on.

The base system will be alike between many various virtual machines, often just the latest release or a (long-term) supported older stable major release of one of the few popular operating systems. Also the applications tend to be at a given moment to consist of the latest release or an older major stable release.

Most software providers only support a limited set of major releases at a given time and have the agreement that they update these major releases with minor updates for bug and security fixes. But within these minor updates they won't provide any new functionality or change any functionality. This way they can promise not to break any of their APIs and all other applications depending on their interface will continue to work, no matter which exact minor revision of their software is used.

Using this knowledge a new possible strategy appears. Because many of the files on a virtual machine are not part of the machine's unique data, but are part of the base system and the applications. If the base system or these applications are often revised, in the second strategy (the file-based one) you end up with many different revisions of each file in the storage database. This limits the possible deduplication efforts, but this limitation could be (partly) avoided if the amount of different revisions were to be reduced. Thus instead of describing files per their unique content (data) that they contain, identifying them by the functionality of these files (which is equal per major revision) is a first step for optimization.

But files are often not alone to provide a set of functionality. In general a collection of files, some executables, library-files, graphical resources and text files make up an application that provides this functionality. Since this method is describing the functionality anyhow, it is easier to only describe the application, instead of the individual files it consists of. But how to know which files are part of which application? Luckily there is already a solution in place on many operating systems. Application's files are centrally managed by the operating system's package manager. Also it has a central register of all applications installed and their version information. Open source operating systems also offer central package repositories from where an installer for a specific application's version can be retrieved. Private repositories for installers of proprietary or custom software distribution can easily be created.

This strategy also depends on the availability of the second strategy (the one that is file-based). Because even though much of the functionality of a virtual machine is within the base-system and the applications' files, the unique data and configuration are still stored within individual files that are not contained in the installer of an application. Also you still need the basic meta-information about the disk, partitioning etcetera for recreating a virtual machine at the target location, just as with the second strategy.

6.5.2 Design

The strategy demands knowledge of the installed operating system, the filesystem structure and access to all the files within the virtual machine to be able to analyze the system. So the design relies just as with the second strategy for detecting the virtual disk image format, analyzing the virtual disk for its partitions, mounting the filesystems etcetera. It is necessary to have a complete virtual filesystem accessible, just as with the second strategy. But instead of analyzing the whole filesystem file-by-file, the inspections starts by finding out which operating system and package management system is used. With this information a list of installed packages can be retrieved from the package management system. After this, the file data which is not to be found in packages has to be stored as was done in the second strategy.
Thus, the analysis of a virtual machine using these steps the results will compile a list of:

- The meta-description of the second strategy, containing the disk size, partition table, boot sector, filesystems and mountpoints.
- The operating system installed and its version and installation location
- A list of applications installed and their version information
- The contents of all configuration files, including a file with the repositories configuration from where the application's installers could be or were retrieved
- All 'data' (files)

These first three sections of this list can be compiled into a 'recipe' for recreating a virtual machine at the target location using the installers of the operating system and the applications as 'ingredients'. After executing the recipe, the two last sections, the configuration files and data files, can be added to the created virtual machine. The great advantage of this method is that these 'ingredients' are universal and would only have to be downloaded once at the target location while they could but reused many times for the recipes of multiple virtual machine, allowing for easy deduplication.
The major design difference compared to other current functional description methods that are currently used in infrastructure-as-code paradigms, is that it allows wrapping an existing virtual machine into the descriptive code. While other infrastructure-as-code methods only allow to encapsulate new (still to be created) virtual machines within their code.

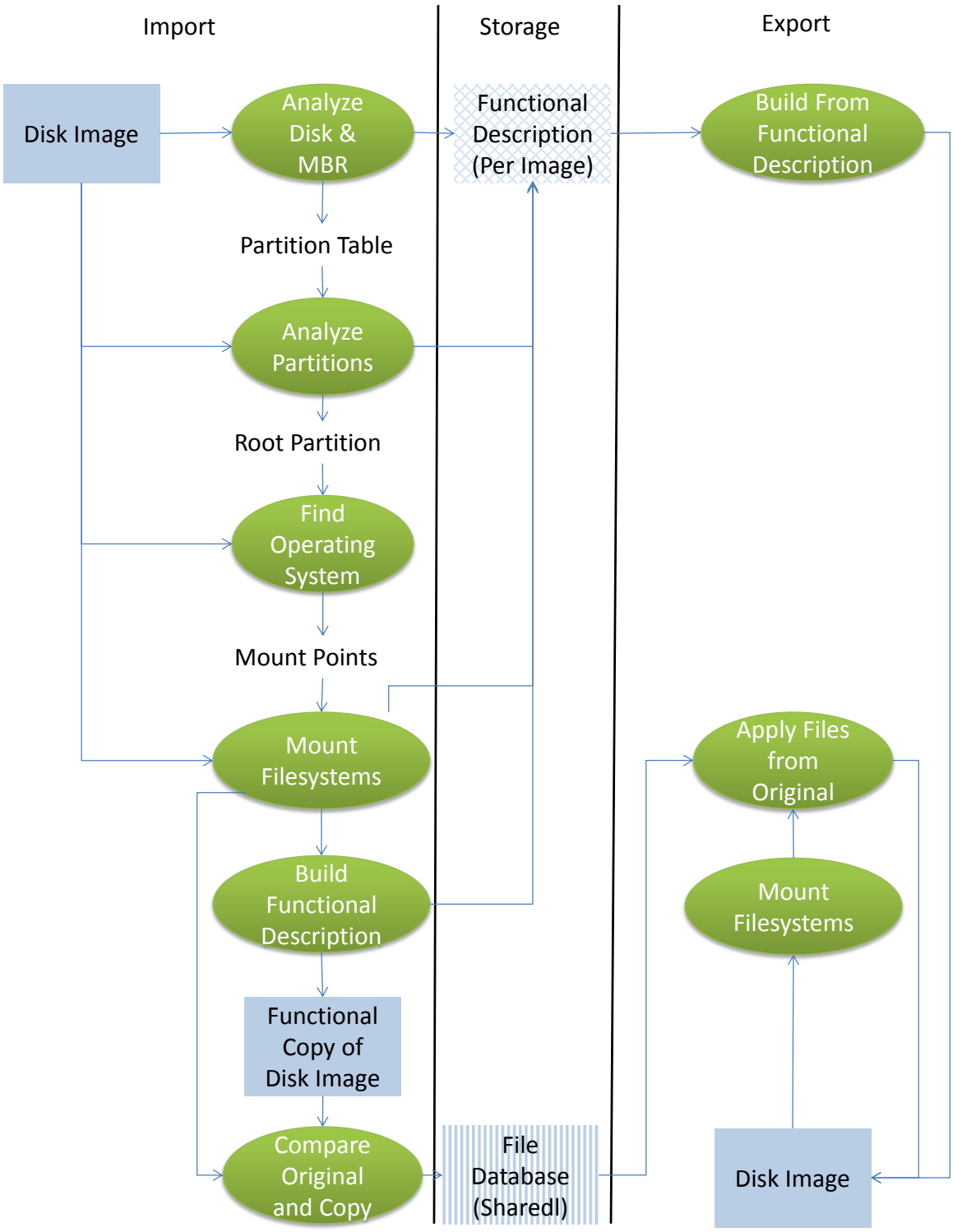

Figure 9 A flow-chart showing the steps that are followed for strategy 3. The disk-image is analyzed for its partition information and the MBR. The partitions are analyzed for their size and their filesystems and mountpoints are retrieved. The filesystems are mounted at their respective mountpoints and a functional description of the virtual machine is generated and stored. Using the functional description, a temporary copy is created. Using the temporary copy the file-differences between the original and the functional copy can be discovered and copy to the

file-database. During reconstruction an image is build using the functional description. The filesystems of this copy are mounted and the files from the database are copied to onto the new image.

6.5.3 Implementation

The prototype for this third strategy is written as Java application that depends, just like the second strategy, on QEMU in combination with libguestfs, the FUSE mount system, the rsync synchronization tool and a bash shell environment. Also it depends on virt-tools and the package management systems that are used within the virtual machine guests.
Just as with the second strategy the prototype starts with creating a libguestfs handle on a virtual machine disk image. Though this time not trough a native libguestfs command-line tool, but from a Java application. Therefore it is necessary to initialize a special libguestfs-java wrapper that connects to the libguestfs-library using JNI.
After creating the handle, again the virtual machine image is assessed for partition information, filesystems and installed operating system. And it is booted using a simple '*supermin*' Linux environment with the KVM (Kernel-based Virtual Machine) virtualization environment. Then a specific class of the libguestfs-library is used by the prototype, the so-called '*virt-inspector*'.
Virt-inspector provides many methods to gather a wide variety of information about the virtual machine guest. The prototype uses these methods to extract the following information:

- The size of the virtual machine's virtual disk,
- The installed operating system and its version,
- A full list of packages installed within the guest machine.

Using this information the prototype uses the virt-builder command line tool from libguestfs to create a new virtual machine. The collected information is passed on as specific parameters to virt-builder, resulting in the creation of a virtual machine with the same operating system version installed and all packages as that were on the original virtual machine. After the creation of this new virtual machine, both the original and the newly created one are mounted on the host with guestmount, as described in the second strategy. Then the differences between the files in the 'data' and configuration directories as they are typically found in the Filesystem Hierarchy Standard, the common directory structure and mapping in Posix-compatible operating systems, are analyzed by rsync. Rsync writes all these changes to an output file that can easily be used to apply the found differences on one of the virtual machines.
Using this method, the original virtual machine is effectively stored within a small commandline functional describing a virt-install, in combination with the output file from rsync. Using these two parts the original virtual machine can be relatively easily recreated at a target location using an operating system and package installers as extra (but reusable) ingredients.

6.5.4 Usage of Prototype

The prototype can be executed by importing vm-analyzer the Java-project in Eclipse. All Java dependencies can be automatically resolved using Maven's dependencies management. Though the libguestfs java-bindings will have to be set manually as a JNI dependency.

In the Java-class BuildFunctionalCopyRecipe three static Strings can be defined. One string ‘source’ with the filepath to the original virtual machine disk image. A ‘target’ for the filepath to the intermediate (reproducible) result that is created form the recipe. And a filepath for ‘targetdiff’ that will store the file-based differences as is in line with the second strategy. If the application will be implemented beyond a prototype, such parameters should be settable using an interface.
The VirtBuilder will first create a VirtBuilderFromVM-object which is passed the ‘source’ and ‘target’ parameters. VirtBuilderFromVM then creates a GuestFS-object and adds the ‘source’ virtual disk image to this object. Then the GuestFS-objecte is launched, which spawns a libguestfs-process using JNI.
Using the libguestfs-process the (root) partitions, filesystems, mountpoints, operating system and a list of installed packages is retrieved. This information is stored as a functional description. This functional description can then be used to compile a complete virt-builder command using the found information as parameters, e.g. a simple version could be:

```
virt-builder fedora-20 --size 20G --arch amd64 --hostname example \
--install apache,mysql --firstboot /tmp/firstbootscript.sh
```

This command is executed using a command shell, which will create the virtual disk image at the filepath location of ‘target’.
After the creation of this new virtual machine image from the functional description, a second Java-object is created from the BuildFucntionalCopyRecipe: MakeFileDelta. MakeFileDelta gets all three arguments passed and mounts both the ‘source’ and the new ‘target’ virtual disk images using guestmount. This is done through the following shell commands:

```
guestmount –r –a /tmp/targetimage.raw /tmp/mountedtarget/
guestmount –r –a /tmp/sourceimage.raw /tmp/mountedsource/
```

This is done twice, for both the ‘source’ and the ‘target'. When both images are mounted, rsync is executed, storing the differences between the two in the ‘targetdiff’ file. The command that is executed could contain:

```
rsync --write-batch=/tmp/file-differences /tmp/mountedsource/
/tmp/mountedtarget/
```

After executing the prototype, using the functional recipe and the ‘targetdiff’ file can be distributed for re-assembly to a target location. The list of packages would normally be at least around 10 but up to hundreds of entities, but has been reduced here to just apache and mysql for the sake of readability.
A simple but complete functional recipe would consist of:

```
virt-builder fedora-20 --size 20G --arch amd64 --hostname example \
--install apache,mysql --firstboot /tmp/firstbootscript.sh

guestmount –r –a /tmp/newimage.raw /tmp/mountedimage/

rsync --read-batch=/tmp/file-differences /tmp/mountedimage/
```

Which would successfully create a functional plus file-level copy of the virtual machine.

6.5.5 Limitations

As many tools from the second strategy implementation are also used in this strategy, many of the limitations are shared. Again the prototype works only for Linux hosts and guests. Security and administrative overhead is necessary to be able to use the KVM virtualization for libguestfs. Also the prototype only functions for Linux distributions which are supported and installable via virt-install.

Libguestfs also had some bugs in its Java-bindings, which were found during the development of the prototype. These bugs are now resolved after interaction with the libguestfs developers, but are still upstream and not available in the major Linux distribution's stock packages. Thus manual installation of an updated libguestfs is necessary.

Also the prototype is at the moment only focused on local application of the strategy mechanisms, no network functionality has been implemented.

6.6 Analysis

A full analysis of the performance each of the different strategies can be considered very hard. There are many varieties of possible virtual machine images as input, but also many possibilities for the given context. Like which distribution mechanisms are actually in place for the delivery of the content, but also the geographical spread and expected use case of the virtual machines. The three scenarios introduced in chapter 4 are meant to cover a vast array of this very variable input, but many aspects can only be theoretically approached with lack of testing availability. Therefore a more theoretical analysis is performed in appendix A, but within this chapter there is chosen to analyze only on the performance that can be directly judged. It is also import to note that that all judgment of the factors is done in a relative manner, the performance of all strategies are always compared with one another.

6.6.1 Support and scope of VM variations

When comparing the performance and scope of all VM variations, it is obvious that strategy 1 scores best. Being completely agnostic to any kind of content that is stored within its cluster-based data-unit, it supports all possible kind of virtual machine image content. Runner up is strategy 2, using files on filesystems, most virtual machine image content will be supported. Though it is possible that in certain situations the partition-information or filesystems cannot be supported by the VMDN, such situations will be very rare. Worst score on this performance factor is for strategy 3. Dependence on knowledge and specific support for the virtual machine's installed operating system, package management and applications can be troublesome if non-mainstream solutions are used.

6.6.2 Deduplication of the Operating System

Deduplication of the content that is part of the operating system is easy to judge for strategy 3: if any supported operating system is used, the deduplication will be very effective. Many megabytes of the operating system data can be reduced for a single virtual machine to a character string of at most several kilobytes.
Strategy 2 will also be able to deduplicate operating system data, since files that are part of the operating system are often relatively static (not changed by the user) and tend not to be changed too much between each operating system's new release. Unfortunately, there are many files that make up the operating system as a whole, so much more meta-information has to be stored for each individual file compared to strategy 3. Therefore this strategy scores second best on this performance factor.
Strategy 1 is least suited for deduplication of the operating system. Even though it also profits from the fact that most operating system information is static, the data stored in clusters is not guaranteed to be stored in the same order on disk. Thus unfortunate mismatches, for which clusters are stored together and hashed, can happen. Also, the operating system is often many megabytes large, while each cluster is only a small amount of data. This means relatively a lot of meta-data has to be stored in comparison to the amount of data that makes up the operating system.

6.6.3 Deduplication of the Applications

For deduplication of the virtual machine image content that constitutes the applications, the same applies as for the operating system for the very same reasons. Strategy 3 would

be, if applicable, the best solution. The second strategy is runner-up, and strategy 1 is the least effective of the three.

6.6.4 Deduplication of the 'User Data'

Deduplication of the content that makes up the 'User Data' is done best by strategy 2. Since all 'User Data' is stored as files, any deduplication can most effectively be done by the strategy that operates at this level, thus strategy 2. Strategy 3 is not applicable at all, since 'User Data' is not within the scope of the functional description of the virtual machine image's contents. Strategy 1 results are less certain. If 'User Data' is (in binary-sense) very repetitive over different virtual machines, or repetitive within one virtual machine, it can be effective. But in many day-to-day situations, the 'User Data' will be varied, and as such strategy 1 does not provide any benefits to this performance factor.

6.6.5 Deduplication of Revision Information

If a virtual machine image's content changes over time, revision information be very useful. It can reduce the amount of data necessary to reassemble any current data if historical data is available. Revision information can be available at all levels, which clusters were changed, which files were changed or which functional aspects have been changed. Depending on the context, any of the strategies can therefore be acting on the most efficient level to pack the changes. And also the actual impact can differ greatly per given situations. For this performance aspect we therefore don't give a score in this chapter.

6.6.6 Speed of importing VM image

For this performance factor, strategy 2 and 3 score best. For a typical virtual machine image of around 2 GiB, a Linux-based operating system with 1GiB of installed packages and 1GiB of User Data, this takes up from 5 to at most 15 minutes. This is because the amount of data to be written is limited and can be stored in efficient manners for file-storage that are already available. Strategy 1 needs 10 to 30 minutes for such an image, performance mostly impacted by the small size of clusters, the very large amount of clusters to be processed and the fast degrading performance when the HashMap database, storing the clusters, has to be reordered when inserting new data.

6.6.7 Speed of reassembling VM image

For the speed of reassembly strategy 1 scores best of the three. Reassembly of the previously described typical 2 GiB virtual machine can easily be performed within 5 minutes, which is possible by the linear writing of data to the disk image during the reassembly. Strategy 2 takes longer, though it also only has to write data to disk, the data is not per se in a linear order. Of course depending on the exact contents, the time necessary to reassemble the disk can take from 10 minutes up to half an hour. Strategy 3 takes longest to reassemble, this because the transactions to disks are a sequence of not only writing, but also reading and manipulating data on the disk. Also CPU time is necessary to process the installation of packages. Because of this, a simple virtual machine image of 2 GiB can take 10 minutes to assemble, but if the packages to be installed are using a lot of complex processing, an image of the same size can take also an hour or even more.

6.6.8 Minimal size for effective cache

This performance factor was not measured within the experiments. In theory one could argue that for each higher level in the virtual machine image's structure, the building blocks become more common. The more common a building block is, the more effective it would be to put in cache. If a building block is only used once, caching it has no benefit. If it is only used rarely, the benefit is limited. So the hypothesis is that the third strategy could benefit most here, but more research on the exact distribution-curve of data-elements is necessary.

6.6.9 Re-usability of data-objects

This performance factor was also not measured within the experiments. Combined with the paragraph above, where it is argued that within the higher level of the virtual machine image's structure more common objects can be found. Which allows for re-usability of data-objects when reassembling various virtual machine images throughout time. Thus higher re-usability is more probable to be found within the second and third strategy. Also strategy 3 allows for re-usage of different versions of the installation packages used. This gives the idea that the third strategy could benefit the most, but more data would have to be collected to research this performance factor for further investigation.

6.7 Results

It is apparent from our analysis that the best strategy can differ greatly depending on the context. In situations where one strategy excels, another might perform horrible, but when the context changes it might be the reverse. Also it might appear that neither strategy really suits the context as an optimal solution.

The context can in general be classified within two aspects: the first one is about what kind of content does the virtual machine that has to be distributed consist of; the second one is for which performance factor(s) are deemed most important in the specific distribution scenario. It is apparent that changing one of these aspects can greatly influence the performance of any strategy.

When applying these aspects on the identified scenarios, various weak and strong points pop-up in relation to the strategies. For example, for scenario 1, the Academic Research Publication, the contents of the virtual machines will very often consist of a popular operating system, most installed software packages will be common and User Data will be unique per publication. Combined with fact that the speed of both importing and reassembly of the virtual machine image are not considered to be of high importance for the use cases, but efficient data storage by re-using common components are. This results in the third strategy to be considered the best candidate for this given scenario, scoring well on all relevant aspects, while its weak points like slow reassembly of the image to be of low importance.

For the second scenario, the Specialized Database Server, the contents of the virtual machine will be uncommon, disqualifying the third strategy completely. For any deduplication and gaining efficient distribution it is hard to point out without a more detailed study whether the first or second strategy would give the best results.

The third scenario, the WebRTC TURN Server, its contents will be probably very common and there would be very little User Data. So the third strategy would be very efficient in means of storage and distribution. But the relatively long time it takes to assemble a virtual machine in the third strategy, would be not compatible with fast, timely response that is demanded from such TURN Servers, disqualifying the third strategy on this crucial aspect. So the hypothesis would be that the second strategy would probably perform best here, being able to reuse many of the files combined with a faster virtual machine assembling mechanism.

7 Future work

7.1 Improving the prototype

One of the most obvious possibilities for future work is improvement of the prototype. Improving the prototype would allow for further and more detailed testing on more scenarios, possibly resulting in more performance factors to be found.
The possible improvements on the prototype could be divided in three possible sets:

- Functionality
- Compatibility
- Performance

The functionality of the prototype is currently limited for strategy 2 and 3. Features like a full integration with a generic file-deduplication storage database is not available in the current revision. Best would be to finalize those features that are relatively easy accomplished by integrating with the current state of the art technologies that are available. Also the development of a content-delivery server-prototypes, that would focus only on managing caches for the clients, could be developed. Many functionalities could easily be implemented by integration with tools like apt-proxy, which can smartly manage mirrors to cache software packages for Debian and derivatives.
Functionality improvements could also be achieved by using new approaches like the usage peer-to-peer to share the data-objects that are used to reconstruct the virtual machines between physical machines.

Compatibility could be improved by supporting more filesystems, operating systems and package-system formats. These improvements could be achieved by further developing the used third party tools like libguestfs and supermin. But some of these are also only possible by improving the prototype itself. E.g. the meta-data that is collected for current partition tables could be adapted to not only support the current (soon to be legacy) Master Boot Record (MBR) method, but to also support the new GUID Partition Table (GPT). This improved compatibility would make strategy 2 and 3 support more possible virtual machine configurations.

The performance improvements could be achieved in two different ways. Or the design of the strategies could be further optimized. E.g. using snapshots on functional copies of virtual machines that are alike might improve the reconstruction time of virtual-machines by re-using these snapshots if configurations reoccur.
A second method for gaining performance improvements might be by rewriting the prototype from Java to another programming language like C++. Java was currently chosen because of previous experience with this programming language. But Java has its limitations for these kind of applications and is probably not the best-suited programming language. Also most third party tools used were written in C++ and offer more application programming interfaces in that language.

7.2 Build a real application around the VMDN

A second opportunity for future work for the VMDN revolves around a direct application of the VMDN's features and best strengths. An analogy can be made with one of the world's current largest CDNs: YouTube [84], [85].
YouTube is not directly advertised to its users as being a CDN (unlike many CDN-specialized businesses). But in the end they run one of the largest CDNs in the world to provide video-streaming services to their customers [86].
The same opportunity might be available for the VMDN. Instead of offering a technical service to distribute pre-created not-to-be-modified content like CDNs do, a service to distribute end-user created content (ergo virtual machines) in a normalized form to other end-users could be offered: YouVM!
YouVM could be a service that allows any user to perform any kind of computer activity within a virtual machine and share this with other users.

First it is important to further look into the analogy of YouVM with YouTube. YouTube had several issues to overcome:

- End-users lack technical knowledge of video (encoding, pre-processing etc.).
- End-users record video in many different video formats and resolutions.
- End-users expect any video to playback in their browser without problems.

For YouVM alike issues apply:

- End-users lack technical knowledge of virtual machines (hypervisors etc.).
- End-users run virtual machines in different virtual machine disk formats, filesystems, operating systems etc.
- End-users would expect to be able to run any shared virtual machine.

How did YouTube handle these issues and what is possible for YouVM? Part of the answer is to be found in re-asking the question in chapter 2.1:"*what is a copy?*".
YouTube recognized that for the end-user that sharing a pixel-perfect copy is not the goal. The end-user just wants to share a video, not pixels. This might not be great for 100% of the scenarios, as sometimes a user would want to share a high-quality video with fine-grained details. But in daily practice it suffices for the vast majority of the users and solves all their problems by converting to users' video file-input to anther video playback-output.
Thus YouTube found the following answers to their questions:

- YouTube created a simple upload mechanism for end-users. The end-user uploads any kind of video file format and resolution, they handle all the backend difficulties that are implied for further distribution.
- YouTube converts and encodes the video in their own internal high quality version. This will be the new 'source' material of the video, not the file that was uploaded.
- YouTube only caches videos in their pre-configured resolutions and encodes them in universal formats that are supported by all major browsers and plug-ins.

So the to-be-created YouVM should support the following:

- It should allow for a simple upload mechanism that accepts any kind of virtual machine disk image format, operating system, filesystem etc.

- It should convert the virtual machine to a standardized format. This might imply changing lower structural layers, like filesystems, or changing the version of the installed software-packages used, allowing for better distribution.
- The virtual machine should be offered in a (set of) format(s) that is universally accepted by most hypervisors like Open Virtualization Format (OVF) and should be directly executable (not possible with current OVF implementations).

By implementing YouVM a real applicable example for usage of the VMDN can be shown. Also, by taking control of the virtual machine's distribution context, just like YouTube does, it enables the leverage for using the VMDNs most efficient strategies.

8 Conclusion

The conclusion of our research is about answering the original research question; whether a Virtual Machine Delivery Network (VMDN) can be more bandwidth efficient for distributing virtual machines than through traditional CDNs. The answer to this is: yes, it can be more efficient, but how much greatly depends on the chosen strategy and on the applicable context.

The answer derives from the answers to the sub-questions. Already in chapter 3.3 it became apparent that bandwidth efficiency optimizations for the distribution of traditional (bitwise identical copies) of various virtual machines is non-existent. If no older revision of a virtual machine is available at the target location (which would allow for usage of difference-only synchronization techniques like rsync) the worst case scenario applies: the transfer of all bits (possibly through a compression filter) for the greatest network distance possible; from source to target location. Traditional CDNs are no answer because they can't take advantage of overlap between the various virtual machines and the rate at which an individual virtual machine would be delivered is too low for effective caching.

The bandwidth efficiency can be improved by using a prototype VMDN solution. For this prototype three possible virtual machine distribution strategies were identified. The first strategy, introduced in chapter 6.3, uses a cluster-based approach, which enables bitwise identical copies of virtual machines. The second strategy, introduced in chapter 6.4, is file-based and creates virtual machine copies only identical to the file level. The third strategy, introduced in chapter 6.5, is functionality-based and creates virtual machine copies that are only identical in exposed functionality. This third strategy also has to rely on the second strategy for the replication of User Data that is not encompassed within functional descriptions.

For the evaluation of the strategies three different scenarios were created. Using these scenarios various performance factors were extracted. The three scenarios were chosen to be varied in such a way that potential weaknesses on different performance factors could be exposed during analysis. During the analysis it became apparent that for per scenario a different strategy would be able to give the best results. Strategies' performance can apparently greatly differ per given context.

The bandwidth efficiency with a VMDN to distribute bitwise identical copies of a virtual machine can already be much higher by using the first strategy compared to traditional distribution methods. But only if an equal virtual machine with the same operating system revision and filesystem is requested repeatedly, for which in the best case scenario up to 80% of the data blocks can be deduplicated using the first strategy which was suggested by literature and also confirmed by the prototype [50].
When virtual machines would downloaded on a frequent basis, the initial download of the 'popular' blocks will in the end be relatively negligible and thus only 20% of the bits would still have to be downloaded per virtual machine. Of course overhead also would have to be added on top of this. An extra possible advantage could be that when

requested clusters are not locally available, they could be cached at host more nearby than the original source, which would allow for even higher bandwidth efficiency.

The bandwidth efficiency for using VMDNs to distribute non-bitwise copies of a virtual machine can be even more efficient. Unfortunately such copies are not always possible, since inner knowledge of the virtual machine is necessary. Deduplication at file-level is applied with the second strategy. Earlier downloaded files can be re-used, which allows in the best case scenario, where a virtual machine has the exact same contents as an earlier retrieved instance, to only demand around 1% of the original image size to describe all file-contents. Which is the relative size that rsync needs to describe the contents of a whole filesystem from the experiment in batch-mode. For virtual machines that are not similar, this percentage would grow equally with the dissimilarity.

By using the third VMDN strategy, for which virtual machine support is even more constrained to certain operating systems, an extreme bandwidth reduction is possible. If the virtual machines are downloaded frequently, the initial download of the operating system's template, package installers and other files only equal around the size of a single virtual machine. But after this initial download each comparable virtual machine (the same operating system or software packages) will barely generate any new bandwidth consumption. Of course these initial downloads don't have to be from the source location, but can be from a server closer to the target. As extra benefit the ratio of which files and packages are installed on a virtual machine is also favorable, with a small set of packages accounting for most of the installations.
One of the disadvantages with this strategy are that for the generation of a non-bitwise copy of a virtual machine that the template and packages also have to be downloaded at the source location. Which makes this strategy only beneficial if these are already available at the source location, or if a copy of a virtual machines is at least downloaded more than twice to a target location. Another disadvantage is that the reconstruction of a virtual machine image can take quite some time. Since the installation of all the packages have to be done through execution of the installer scripts. These scripts take processing time and the transactions to the disks are in general not linear.

Appendix A: Analysis of Strategies

All three strategies have been tested in a limited experiment with the prototype VMDN implementation. Various virtual-machine guests were created to test the various performance factors. Some (parts of) the strategies could be performed in an automatic manner, like the whole first strategy. But e.g. the second strategy was only performed by manually using the appropriate shell commands. Also, for the assessment of e.g. the first strategy more and a multitude of data was available from the literature than from our single implementation. Also these other researches sometimes had implemented more advanced techniques like finding the best cluster-offset per image. Therefore the numbers from literature can take precedence when judging the strategy performance.

The analysis of all three strategies will be discussed per strategy in this appendix. For each strategy first a table-overview is presented with the performance factors and their score, including a small note for each score. After that each performance factor is more explicitly discussed and be given an one of the five qualitative scores, ranging from double minus (--) to double plus (++). Where possible this indicative score will be further quantified for expected typical contexts for virtual machines. Any exact number for the score is hard to qualify, since virtual machine guests can vary a lot. But these numbers are chosen for most typical are virtual machines: having a single virtual disk, at most three disk partitions, no logical volume management, a single operating system, no disk or filesystem encryption and a disk size ranging from 2 to 20 gigabytes. For the host machine a physical traditional spinning hard disk is assumed, plenty of RAM (Random Access Memory) and a CPU (Central Processing Unit) found in typical server-hardware.

After these performance analyses per strategy an analysis for the set of applicable scenarios will be performed. Indicating which strategies would be best suited to each scenario.

Strategy 1: Clusters

Strategy 1, the cluster-based approach, is well known the academic world and thus it advantages and disadvantages are also partly known from multiple experiments.

Performance factor	**Performance Score**
Virtual Machine Characteristics	
Support and scope of VM variations	++ (agnostic of content)
Deduplication of the Operating System	+ or -- (depending on exact matches or not)
Deduplication of the Applications	-- (easy mismatches of versions)
Deduplication of the ‘User Data’	-- (with exception of exact copies)
Deduplication of Revisions	-- (often impractical)
Distribution Factors	
Speed of importing VM image	+/- (10 to 30 minutes)
Speed of reassembling VM image	++ (less than 5 minutes)
Minimal size for effective cache	-- (easily gigabytes to be useful)
Re-usability of data-objects	-- (only exact matches)

Support and scope of virtual machine variations
Strategy 1 is agnostic of its content. The technology only interacts with the disk clusters from the virtual disk device. Also all techniques are from the virtual machine's guest perspective like it acts even lower than the hardware interface. Thus on this aspect the support and scope for various virtual machines, irrespective of the operating systems and filesystems used the score is 'perfect': ++

Deduplication of the Operating System
Strategy 1 allows for deduplication of the operating system content. It is important to notice that the operating system is the first thing installed on a (virtual) system and (as earlier described) mostly automated process of scripts and installers. This allows for a relatively high deduplication score: +. But it is important to remark that this only applies for situations where the operating system and its revision are exactly the same. Luckily some of the context is known: According to W3Techs the most popular operating system for servers is Linux with around 65% market share [87]. On virtual machines this can be expected to be even higher, since it simplifies license issues compared to competing operating systems. Then the two largest Linux distributions for servers are Debian and CentOS with both around 30% market share. The runner-ups are Ubuntu and Red Hat with both around 15% market share. These numbers suggest that around 90% of all operating systems on Linux servers are served by just four different distributions. If we further split out the details for a typical distribution, using Debian's Popularity Contest, it is visible that i386 and amd64 are by far the most popular platform architectures [88]. And when a new major revision of the distribution is released, there is a relatively fast uptake of this new revision. It quickly moves to be the dominating revision in general and certainly is the primary choice for any new installations.
Since it is known that the deduplication only works well for a single revision of a single operating system distribution it is important to note that this good score directly degrades to a bad score as soon as it concerns non-popular operating systems, an outdated version or on an uncommon architecture. (A typical IaaS provider should be able to manage this context to its advantage for this strategy by only providing a select choice for all of these three factors to its customers when creating and installing new virtual machines.)

Deduplication of the Applications
Strategy 1 is not so suitable for deduplication of applications. Even though deduplication can still be applied on the data stored within the clusters, the nature of applications is that they can be installed in any order. This reduces the amount of linear overlap of the data in the clusters between different virtual machines. Also small minor differences can exist between the applications on various virtual machines. This can be differences of different minor revisions, or certain patches applied in specific versions but also the usage of different compilers or compile flags. Even if these differences are small they can create a complete mismatch for deduplication of the data in the clusters. The score for this performance factor is thus low: --.

Deduplication of the 'User Data'

Strategy 1 is not so suitable for deduplication of 'user data'. Even though deduplication can still be applied on the data stored within the clusters, it is unlikely that the 'user data' is written in the same linear patterns, decreasing the likelihood of matches. And it would in any case only apply for the situations where multiple virtual machines have exact identical common 'user data'. If no 'user data' is common between virtual machines or it has some differences between them, this method doesn't provide any gains. The score for this performance factor is thus low: --.

Deduplication of Revision Information
Strategy 1 is in general unsuitable for deduplicating content that has multiple revisions. If a data object changes at random locations, various clusters could change and no match for deduplication is possible anymore.
The exception is for content that never changes at random locations but only consistently within a single or a few clusters, while the rest of the used clusters are high in number and all static. Or if only data is added to the end of a data object for each new revision and all the old data is static. The first situation is rare and as such not applicable. The second situation is only possible for certain storage mechanisms, like redirect-on-write. But those mechanisms then imply also having to send over all discarded (old) data. So the score on this performance factor is in general bad: --.

Speed of importing VM image
The speed of importing a virtual machine image is quite reasonable for strategy 1. The image is read linearly from start to end. If the image is not fragmented on the disk this should go quite fast. Calculating a hash for each cluster also goes relatively fast, but adding new clusters to a database containing already many hashes is slow. Or the database has to grow, which will give a performance hit once a while when the database has to be re-ordered, or old entries have to be deleted, which requires a mechanism that detects the least used (unpopular) entries and balances the database. The score for this performance factor is +/-.

Speed of reassembling VM image
The speed of retrieving a virtual machine image is good for strategy 1. All the hashes are linearly retrieved, the database with clusters is ordered on the hash and can find the data-objects fast. Writing the data-objects to disk is a linear process and the total size is known in advance, so the necessary disk space can already be reserved by the filesystem. This makes this performance factor 'perfect': ++.

Minimal size for effective cache
Unfortunately strategy 1 relies on many very small data-objects. Thus the database has to contain many objects. Also the curve for matching identical clusters is not ideal. If ignoring the clusters filled with zeroes (empty disk space or other unused clusters) the distribution is a relatively equal spread of almost all obtained clusters if you have a bare virtual machine installation (without applications and 'user data'). It is not rewarding to create a database if you wouldn't be able to cache at least all these clusters, implying the need for a large database to be effective: --.

Re-usability of data-objects
All data obtained for clusters can only be used for matching hashes. Matching clusters only appear (as discussed earlier) mostly for exact identical operating system distributions of a specific version and architecture, the re-usability is low. No replacement of one cluster for the other is possible. Meanwhile the data-objects are very small in size and overhead for matching and retrieving will be relatively large.

Strategy 2: Files

Strategy 2, the file-based approach, is a combination of current file syncing and deduplication techniques with know-how of the virtual machine structure. Of both techniques empirical data is available about the performance.

Performance factor	**Performance Score**
Virtual Machine Characteristics	
Support and scope of VM variations	+ (all major filesystems and OSes)
Deduplication of the Operating System	+ (file-based, difference-only possible)
Deduplication of the Applications	+ (file-based, difference-only possible)
Deduplication of the 'User Data'	++ (file-based, difference-only possible)
Deduplication of Revisions	+/- (difference-only, not functionality)
Distribution Factors	
Speed of importing VM image	+ (5 to 15 minutes)
Speed of reassembling VM image	+/- (10 to 30 minutes)
Minimal size for effective cache	+ (very flexible)
Re-usability of data-objects	+/- (difference-only, not functionality)

Support and scope of virtual machine variations
Strategy 2 has to understand the files on a virtual machine's disk. For this it has to understand virtual disk images, disk layouts, partition tables, filesystems, possibly LVM (logical volume management), mountpoint configurations etcetera.
Luckily, of all these aspects there are a limited set of choices. Only a couple of virtual disk image formats exists, disk layout is universal, for filesystems there are multiple choices but only a few are popular, for LVM there is a limited set of technologies and mountpoints are according to Windows' standard or the Unix standard. For all of these components there are good (open source) drivers and/or documentation available to implement support.
This results in this performance factor scoring 'good': +.

Deduplication of the Operating System
Strategy 2 is quite good for deduplicating operating system content. Since most files will be identical between different virtual machines as long as they will have the same operating system distribution's version installed, all these files can easily be deduplicated. There might even be a possibility to deduplicate between different versions, since small differences might be just described by a difference-only-description from a common version. The same might apply even with different distributions or platform

architectures, since many files are stored on the same path with the same file name and could easily be matched for alikeness.
Of course, deduplication is not possible between whole different operating systems. The score of this performance is therefore ‘good’: +.

Deduplication of the Applications

Just as with the operating system all identical files of applications can easily be matched in strategy 2. No trouble for in which order the applications were installed, the path and name of the file will always be consistent and usable. Again alikeness between different revisions and distributions can also be checked if path and filename correspond and stored as difference-only-descriptions.
Also the popularity of applications can be indexed per file. Thus allowing for caching more popular or universal components of the application, while keeping optional or diverse files aside. The score of this performance factor is ‘good’: +.

Deduplication of the ‘User Data’

Strategy 2 is relatively suitable for deduplication of ‘user data’. Of course, if the content ‘user data’ is not popular or nothing alike to be found in other virtual machines, there is nothing to be gained. But in case there is any likeness of the ‘user data’ between virtual machines and it’s popular, this strategy should be able to deduplicate it. It can look for identical content and just deduplicate it. Or it can detect identical paths, filenames and try to describe any data by difference-only-descriptions from a common revision. The score of this performance factor is ‘good’: +.

Deduplication of Revision Information

Strategy 2 is in general suitable for deduplicating content that has multiple revisions. It can leverage the knowledge of path and filename to detect common ancestry between files and us it too its advantage. Unfortunately the strategy doesn’t allow for replacing a file with another one with equal functionality, because the file is still assessed on its content and not on its function. Thus differences between a file’s revisions always have to be distributed. The score of this performance factor is +/-.

Speed of importing VM image

The speed of importing a virtual machine image is quite good for strategy 2. The virtual disk image is read file by file, which is not the fastest method since the data is not in-order on the disk. But if a smart filesystem was used the retrieval of the meta-data and the layout of the data on the disk might be quite optimal. Also having files that are larger in size can have a positive impact on the average transfer speed. Adding the files to the database should be fast, since the files are stored separate of the meta-data and the amount of data-objects is limited. The performance factor is ‘good’: +.

Speed of reassembling VM image

The speed of retrieving a virtual machine image is reasonable for strategy 2. First the virtual disk image, the filesystems etcetera have to be created at the target location. All the files can be retrieved relatively fast from a database and if large enough this good also go with reasonable average transfer speed. But when the files are written to the filesystem on the target virtual disk the filesystem has to decide on all the locations, which might be

non-linear. Also all the meta-data on the files will have to be applied. This makes this performance factor +/-.

Minimal size for effective cache
Strategy 2 allows for any size of database. All files, whether they are part of the operating system, applications or 'user data' they can all fairly be judged on their popularity to be cached or not. Also the spread of the curve for the popularity of files is not too equal. Thus it is possible to just cache just the popular instead of all files. The score of this performance factor is 'good': +.

Re-usability of data-objects
As described earlier, sometimes alikeness can be used to cache and reuse some files with just a difference-only-description. And of course identical files can be reused. Unfortunately no re-usage on functionality is possible with this strategy, which limits re-usability between different versions of a file. This performance factor scores +/-.

Strategy 3: Functionality

Strategy 3, the functional approach, is a combination of the second strategy combined with knowledge of the packaging system of the operating system within the virtual machine. Of this strategy the only empirical data is available than from the prototype, but also statistics of the popularity of packages can be used.

Performance factor	**Performance Score**
Virtual Machine Characteristics	
Support and scope of VM variations	- (only certain OSes & package managers)
Deduplication of the Operating System	++ (functional description)
Deduplication of the Applications	++ (functional description)
Deduplication of the 'User Data'	n.a. (uses strategy 2)
Deduplication of Revisions	++ (matching functional descriptions)
Distribution Factors	
Speed of importing VM image	+ (5 to 10 minutes)
Speed of reassembling VM image	-- (10 to 60 minutes or more)
Minimal size for effective cache	++ (very flexible)
Re-usability of data-objects	++ (matching functional descriptions)

Support and scope of virtual machine variations
Strategy 3 inherits all constraints from strategy 2. Only if the strategy has access to the files on the virtual machines it can try to assess the functionality. For the assessment it is necessary that the strategy can understand which operating system is installed and how the packaging mechanism works. Luckily there is a limited set of operating systems available, so detection of it should be possible. But between the various operating systems there tend to be different packaging mechanisms. This could pose serious limits on the strategy for the scope of operating systems that can be supported.

An extra limitation is also that it should be possible to recreate the virtual machine installation at the target location. This demands that the specific operating system and its package mechanism also have to be supported by the virtual machine creation tool that is used at the target location.
This results in this performance factor scoring 'bad': -.

Deduplication of the Operating System
Strategy 3 is ideal for deduplication the operating system. If the operating system can be described by a short identifier like its name, version and a list of installed features, an exact replica can be generated. This means that within a few kilobytes of textual data many megabytes of data can be identified and efficiently deduplicated through re-usage of this data for installations of the operating system on multiple virtual machines.
The score of this performance is therefore 'perfect': ++.

Deduplication of the Applications
If all applications can be described as a list of packages for the package management system, using identifiers like names and versions combined with the knowledge for which operating system and its version they are meant. Then you can summarize a whole virtual machine's many megabytes of installed applications into just a few kilobyte of textual description. The installer packages that are necessary to install the applications with the virtual machine also tend to be smaller than the size the take up when installed, because they are stored in highly compressed archives. And these installer packages can, just like with the operating systems, be re-used for the installation multiple virtual machines.
It is important to note that if an application is not packaged on the virtual machine through the package management system, it should be considered 'User Data' in this context and cannot profit from the gains of this strategy and has to fall back on strategy 2.
The score of this performance factor is 'perfect': ++.

Deduplication of the 'User Data'
Strategy 3 is unable to deduplicate any 'user data' if it is not packaged. It is advantageous to deploy any 'user data' in the form of a package that can be handled by the package management system, if it is completely static over multiple virtual machines. For regular file-based 'user data' this strategy has to rely on the implementation of strategy 2.
The score of this performance factor is here 'not applicable'.

Deduplication of Revision Information
Strategy 3 is suitable for deduplication content with multiple revisions. Since for all the packages a name and version number are known, revisions can easily be tracked. And difference-only description is already available for current package management systems when updating from an older to a current version. Also the strategy gives way to proactively use the information of revisions to recreate equal functionality at a target location why in fact using another revision than the original virtual machine contained. This allows for further optimizations in the distribution process.
The score of this performance factor is 'perfect': ++.

Speed of importing VM image
The speed of importing a virtual machine image is quite good for strategy 3. Once the virtual machine's filesystem is accessible, detecting the virtual machine's operating system and its version can often be done by checking a single file. Querying the package management system for a list of installed application is also often just a process of checking a single or a small set of files. So these steps are fast and the list with functional information can be composed fast. But the strategy still has to perform the same steps as strategy 2 for indexing the file-data that is not identified by the functional description.
The score of this performance factor is 'good': +.

Speed of reassembling VM image
The speed of retrieving a virtual machine image can be slow for strategy 3. The retrieval of the functional description will be fast. But from this functional description the whole virtual machine has to be rebuild locally. So as with strategy 2, the virtual disk image, the filesystems etcetera have to be created at the target location. But also an installation medium or basic image of the operating system might have to be retrieved, depending on the fact if it was already downloaded earlier. Also (some of) the package installers have to retrieved, depending on local availability. Also the 'user data' has to be downloaded. After the downloading all packages have to be sequentially installed, this can consume quite some time, because installers use scripts and often have next to writing new data also modify existing data, which can be relatively slow on a virtual disk. Afterwards the 'user data' still has to be copied to the virtual machine's filesystem as in strategy 2.
This makes this performance factor 'very bad': --.

Minimal size for effective cache
Strategy 3 allows for any size of database. Only popular operating systems and packages and their latest (major) revisions have to be cached. This allows for high hit ratios on the cache while it can be kept compact. Of course the 'user data' is beside the scope of this specific cache.
The score of this performance factor is 'perfect': ++.

Re-usability of data-objects
As described earlier, version information is available and difference-only descriptions are already possible. But the greatest strength is that re-usage on functionality is possible with this strategy. Versions of packages that are close can be replaced with one another, which greatly enhances re-usability. This performance factor scores ++.

Overview
When collecting the performance factors per strategy together into a single table we get the following overview:

Performance factor	**Strategy 1 (Clusters)**	**Strategy 2 (Files)**	**Strategy 3 (Functionality)**
Virtual Machine Characteristics			
Support and scope of VM variations	++	+	-

Deduplication of the Operating System	+ (possibly --)	+	++
Deduplication of the Applications	--	+	++
Deduplication of the 'User Data'	--	++	n.a. (Strat 2)
Deduplication of Revisions	--	+/-	++
Distribution Factors			
Speed of importing VM image	+/-	+	+
Speed of reassembling VM image	++	+/-	--
Minimal size for effective cache	--	+	++
Re-usability of data-objects	--	+/-	++

From this overview already several aspects can be deducted easily: strategy 1 and strategy 3 have almost reversed performance characteristics. The factors where strategy 1 is superior (the support and scope of VM variations, and the speed of retrieving a VM image) are the exact factors that are the worst performers for strategy 3 and vice versa. While strategy 2 holds the middle grounds, that doesn't really excel besides the deduplication of 'user data' but also doesn't have any really weak points.

Apparently there is not a single strategy that is always better than the other. Thus the best performing strategy has to be decided depending on the context. That is why the strategies will be discussed using the earlier scenarios. Per scenario another (set of) performance factors will be (more) important and fit of each strategy will be assessed for that specific scenario.
It is difficult to quantify the exact performance factor's result in combination with the performance factor's priority. But in this research we have chosen to adhere to the calculation mechanisms that are used normally for risk management, because in risk comparable situations arise were multiple influencing factors have to be weighed against together to produce a judgment on a good or a bad result . Risk is normally calculated through the multiplication of Impact and Probability. This same multiplication is also here performed, but then the performance factor's importance as Impact and the strategy's performance score as occurrence. The multiplication is applied as follow: for a low priority the multiplication is zero, for medium it is one and for high it is multiplied with two. For critical importance of a performance factor, a text-based notification is in place if failing this factor.

Scenario 1: Academic Research Publication

Scenario 1 is about a virtual machine containing the tools and dataset use for academic research and accompanies a scientific paper. If the matrix of importance of the performance factor is taken from the scenarios chapter and combined through a process of multiplication of importance with the performance score, the following matrix is the result:

Performance factor	**Strategy 1 (Clusters)**	**Strategy 2 (Files)**	**Strategy 3 (Functionality)**
Virtual Machine Characteristics			
Support and scope of VM variations	+	+	-

Deduplication of the Operating System	----	++	++++
Deduplication of the Applications	----	++	++++
Deduplication of the 'User Data'	0	0	0
Deduplication of Revisions	----	0	++++
Distribution Factors			
Speed of importing VM image	0	0	0
Speed of reassembling VM image	0	0	0
Minimal size for effective cache	----	++	++++
Re-usability of data-objects	--	0	++

From the matrix overview it is clear that the best candidate for this context is strategy 3. It has a positive score on the performance factors that are of high importance. The less scoring performance factors are mostly irrelevant to the context of this scenario.

Scenario 2: Specialized Database Server

Scenario 2 is a virtual machine for performing a set of specific database operations. If the matrix of importance of the performance factor is taken from the scenarios chapter and combined through a process of multiplication of importance with the performance score, the following matrix is the result:

Performance factor	**Strategy 1 (Clusters)**	**Strategy 2 (Files)**	**Strategy 3 (Functionality)**
Virtual Machine Characteristics			
Support and scope of VM variations	++++	++	--
Deduplication of the Operating System	----	++	n.a.
Deduplication of the Applications	----	++	n.a.
Deduplication of the 'User Data'	----	0	0
Deduplication of Revisions	0	0	0
Distribution Factors			
Speed of importing VM image	0	0	0
Speed of reassembling VM image	++	0	--
Minimal size for effective cache	0	0	0
Re-usability of data-objects	0	0	0

From the matrix overview strategy 2 seems to have the best, but still not high, score. It is important to note the remarks for the low deduplication scores. For strategy 1 the problem is that the operating system will be unique and therefor hard to deduplicate. For strategy 3 the problem is also that a non-common operating system and the usage of non-package management applications disable the applicability of the strategy and render it useless.

Scenario 3: WebRTC TURN Server

Scenario 3 is a virtual machine for relaying WebRTC connections. If the matrix of importance of the performance factor is taken from the scenarios chapter and combined through a process of multiplication of importance with the performance score, the following matrix is the result:

Performance factor	**Strategy 1 (Clusters)**	**Strategy 2 (Files)**	**Strategy 3 (Functionality)**
Virtual Machine Characteristics			
Support and scope of VM variations	0	0	0
Deduplication of the Operating System	++	++	++++
Deduplication of the Applications	----	++	++++
Deduplication of the 'User Data'	0	0	0
Deduplication of Revisions	0	0	0
Distribution Factors			
Speed of importing VM image	0	0	0
Speed of reassembling VM image	++++	0	Cannot fulfill critical aspect
Minimal size for effective cache	----	++	++++
Re-usability of data-objects	----	0	++++

From the matrix we see a mixed result. Strategy 2 seems best suitable overall, but doesn't have extremely well scores. Meanwhile strategy 3 seems to score well on most factors, with exception of the most important factor: the retrieval speed of a virtual machine image. This most crucial aspect has a bad score. Strategy 1 on the other hand is fast with retrieving a virtual machine image, but the gains there might be completely negated by not being able to efficiently use a cache or re-using the retrieved data objects.

Glossary

- Bash; The GNU Bourne Again SHell, a commandline shell for Unix/Linux. See www.gnu.org/s/bash/
- CDN; Content Delivery Network, a party specialized in delivering content of their customers to many end-users with high availability and high performance all over the world, using a large distributed system of servers.
- chroot; an operation on Unix/Linux that changes the apparent root directory locking the process and its children in a "chroot jail" that cannot access files outside the designated directory tree
- Hypervisor; computer software on a host computers that creates the environment in which virtual machines can be run.
- KVM (Kernel-based Virtual Machine); a Linux kernel module and popular hypervisor based on QEMU that provides virtualization to Linux host computers. See www.linux-kvm.org
- Lossy Compression; A compression method that encodes data using inexact approximations or discarding of original data, often under the assumption that it does not noticeably negatively influences the function of the data.
- QCoW2; QEMU Copy-on-Write 2 is a format for virtual machine disk image storage. See people.gnome.org/~markmc/qcow-image-format.html
- Rsync; A utility that provides fast incremental file transfer. See rsync.samba.org
- Supermin; A lean Linux-appliance to boot a virtual configuration. It is best comparable to having a virtual machine configuration where supermin is started from a small bootable disk. From the regular disks attached to the virtual machine is not booted (and thus the installed operating system is not started) but is accessible. See libguestfs.org/supermin.1.html
- User Data; The configuration and data that is neither part of the Operating System neither any of the static or data that is part of any Application. In the Linux context this would at least include directories like /etc/, /home/, /root/ and /var/lib/. In general a specific combination of this data is unique to a single virtual machine.
- Virtsync; A fork from the rsync project that allows the usage of rsync --sparse in combination with --inplace, which is normally not supported. See www.virtsync.com
- VMDN; Virtual Machine Delivery Network. A concept derivative of a CDN that has been authored in this research.
- WebRTC; Web Real-Time Communication, an API definition by the World Wide Web Consortium (W3C) to allow real-time browser-to-browser applications without plugins. See www.webrtc.org

Bibliography

[1] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor.," in *USENIX Annual Technical Conference, General Track*, 2001, pp. 1–14.

[2] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: Current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, 2005.

[3] J. E. Smith and R. Nair, "The architecture of virtual machines," *Computer*, vol. 38, no. 5, pp. 32–38, 2005.

[4] G. Orzell, "The Netflix Tech Blog: Auto Scaling in the Amazon Cloud." [Online]. Available: http://techblog.netflix.com/2012/01/auto-scaling-in-amazon-cloud.html. [Accessed: 04-Aug-2014].

[5] J. Ousterhout, "Virtual Machine Monitors." [Online]. Available: http://web.stanford.edu/class/cs140/cgi-bin/lecture.php?topic=vmm. [Accessed: 04-Aug-2014].

[6] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A Break in the Clouds: Towards a Cloud Definition."

[7] P. Mell and T. Grance, "The NIST definition of cloud computing," *Natl. Inst. Stand. Technol.*, vol. 53, no. 6, p. 50, 2009.

[8] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the Cloud? An architectural map of the Cloud landscape," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, 2009, pp. 23–31.

[9] I. Lazar and W. Terrill, "Exploring content delivery networking," *IT Prof.*, vol. 3, no. 4, pp. 47–49, 2001.

[10] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An analysis of internet content delivery systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 315–327, 2002.

[11] H. Yu, D. Zheng, B. Y. Zhao, and W. Zheng, "Understanding user behavior in large-scale video-on-demand systems," in *ACM SIGOPS Operating Systems Review*, 2006, vol. 40, pp. 333–344.

[12] L. A. Adamic and B. A. Huberman, "Zipf's law and the Internet," *Glottometrics*, vol. 3, no. 1, pp. 143–150, 2002.

[13] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 1999, vol. 1, pp. 126–134.

[14] D. N. Serpanos, G. Karakostas, and W. H. Wolf, "Effective caching of Web objects using Zipf's law," in *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on*, 2000, vol. 2, pp. 727–730.

[15] W. B. Norton, "Internet service providers and peering," in *Proceedings of NANOG*, 2001, vol. 19, pp. 1–17.

[16] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet inter-domain traffic," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 75–86, 2011.

[17] K. Nance, B. Hay, and M. Bishop, "virtual machine introspection," *IEEE Comput. Soc.*, 2008.

[18] P. M. Chen and B. D. Noble, “When virtual is better than real [operating system relocation to virtual machines],” in *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, 2001, pp. 133–138.
[19] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao, “Quantitative comparison of Xen and KVM,” *Xen Summit Boston MA USA*, pp. 1–2, 2008.
[20] J. Hwang, S. Zeng, T. Wood, and others, “A component-based performance comparison of four hypervisors,” in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, 2013, pp. 269–276.
[21] E. Scholten, *Enterprise Hypervisor Comparison*. Version, 2012.
[22] B. Pfaff, T. Garfinkel, and M. Rosenblum, “Virtualization Aware File Systems: Getting Beyond the Limitations of Virtual Disks.,” in *NSDI*, 2006.
[23] K. R. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, and H. Lei, “An empirical analysis of similarity in virtual machine images,” in *Proceedings of the Middleware 2011 Industry Track Workshop*, 2011, p. 6.
[24] P. Liu, Z. Yang, X. Song, Y. Zhou, H. Chen, and B. Zang, “Heterogeneous live migration of virtual machines,” in *International Workshop on Virtualization Technology (IWVT’08)*, 2008.
[25] “Tip: Defragmentation | VMware Fusion Blog - VMware Blogs.” [Online]. Available: https://blogs.vmware.com/teamfusion/2008/10/tip-defragmenta.html. [Accessed: 30-Jun-2014].
[26] C. Tang, “FVD: A High-Performance Virtual Machine Image Format for Cloud.,” in *USENIX Annual Technical Conference*, 2011.
[27] R. Strijkers, R. Cushing, D. Vasyunin, C. de Laat, A. S. Z. Belloum, and R. Meijer, “Toward Executable Scientific Publications,” *Procedia Comput. Sci.*, vol. 4, pp. 707–715, 2011.
[28] G. Motika and S. Weiss, “Virtio network paravirtualization driver: Implementation and performance of a de-facto standard,” *Comput. Stand. Interfaces*, vol. 34, no. 1, pp. 36–47, 2012.
[29] R. Russell, “virtio: towards a de-facto standard for virtual I/O devices,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 95–103, 2008.
[30] M. Hagström, “Remote desktop protocols: A comparison of Spice, NX and VNC,” 2012.
[31] B.-C. Bjork, A. Roos, and M. Lauri, “Scientific Journal Publishing: Yearly Volume and Open Access Availability,” *Inf. Res. Int. Electron. J.*, vol. 14, no. 1, Mar. 2009.
[32] A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan, “Webrtc 1.0: Real-time communication between browsers,” *Work. Draft W3C*, vol. 91, 2012.
[33] E. A. Shriver, C. Small, and K. A. Smith, “Why does file system prefetching work?,” in *USENIX Annual Technical Conference, General Track*, 1999, pp. 71–84.
[34] D. Hildebrand, A. Povzner, R. Tewari, and V. Tarasov, “Revisiting the Storage Stack in Virtualized NAS Environments.,” in *WIOV*, 2011.
[35] D. Naik and L. L. C. Niriva, “Virtual Machine Storage—often overlooked optimizations,” in *2010 Storage Developer Conference, Storage Networking Industry Association*, 2010.
[36] R. Timo and Y. Tatu, “scp - FreeBSD.” [Online]. Available: http://nixdoc.net/man-pages/FreeBSD/scp.1.html#HISTORY. [Accessed: 01-Aug-2014].

[37] J. Postel and J. Reynolds, “File Transfer Protocol.” [Online]. Available: http://tools.ietf.org/html/rfc959. [Accessed: 01-Aug-2014].
[38] D. Giampaolo, *Practical file system design with the Be file system*. Morgan Kaufmann Publishers Inc., 1998.
[39] Samba Team, “rsync.” [Online]. Available: http://rsync.samba.org/. [Accessed: 01-Aug-2014].
[40] C. Dew, “virtsync - the solution to rsync not supporting --sparse with --inplace.” [Online]. Available: http://www.virtsync.com/. [Accessed: 01-Aug-2014].
[41] B. Khasnabish, W. Jin, and M. Li, “Content De-duplication for CDNi Optimization,” 2013.
[42] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, “Globally distributed content delivery,” *Internet Comput. IEEE*, vol. 6, no. 5, pp. 50–58, 2002.
[43] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki, “Improving duplicate elimination in storage systems,” *ACM Trans. Storage TOS*, vol. 2, no. 4, pp. 424–448, 2006.
[44] P. Kulkarni, F. Douglis, J. D. LaVoie, and J. M. Tracey, “Redundancy Elimination Within Large Collections of Files.,” in *USENIX Annual Technical Conference, General Track*, 2004, pp. 59–72.
[45] J. S. YADAV and M. Y. A. JAIN, “Distributed File System,” *Int. J. Sci. Res. Educ.*, vol. 1, no. 06, 2014.
[46] R. Sharpe, “Just what is SMB?,” *Oct*, vol. 8, p. 9, 2002.
[47] J. Barreto, L. Veiga, and P. Ferreira, “Hash challenges: Stretching the limits of compare-by-hash in distributed data deduplication,” *Inf. Process. Lett.*, vol. 112, no. 10, pp. 380–385, 2012.
[48] J. Tate, F. Lucchese, R. Moore, and S. A. N. TotalStorage, *Introduction to storage area networks*. IBM Corporation, International Technical Support Organization, 2005.
[49] A. T. Clements, I. Ahmad, M. Vilayannur, J. Li, and others, “Decentralized Deduplication in SAN Cluster File Systems.,” in *USENIX Annual Technical Conference*, 2009, pp. 101–114.
[50] K. Jin and E. L. Miller, “The effectiveness of deduplication on virtual machine disk images,” in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, 2009, p. 7.
[51] D. T. Meyer, G. Aggarwal, B. Cully, G. Lefebvre, M. J. Feeley, N. C. Hutchinson, and A. Warfield, “Parallax: virtual disks for virtual machines,” in *ACM SIGOPS Operating Systems Review*, 2008, vol. 42, pp. 41–54.
[52] I. LANGOUEV, P. Lu, A. V. Pershin, S. R. Piduri, and E. Weathers, *Tracking block-level changes using snapshots*. Google Patents, 2012.
[53] A. Mashtizadeh, E. Celebi, T. Garfinkel, M. Cai, and others, “The design and evolution of live storage migration in VMware ESX,” in *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, 2011, pp. 14–14.
[54] K. Takahashi, K. Sasada, and T. Hirofuchi, “A fast virtual machine storage migration technique using data deduplication,” in *CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization*, 2012, pp. 57–64.

[55] “VMware KB: Changed Block Tracking (CBT) on virtual machines (1020128),” *VMware Knowledge Base*. [Online]. Available: http://kb.vmware.com/kb/1020128. [Accessed: 30-Jun-2014].
[56] R. W. M. Jones, “virt-sparsify.” [Online]. Available: http://libguestfs.org/virt-sparsify.1.html. [Accessed: 30-Jun-2014].
[57] G. Ammons, V. Bala, T. Mummert, D. Reimer, and X. Zhang, “Virtual machine images as structured data: the mirage image library,” *Proc. USENIX HotCloud*, 2011.
[58] “IBM Mirage Project.” [Online]. Available: http://www.research.ibm.com/mirage/.
[59] D. T. Meyer and W. J. Bolosky, “A study of practical deduplication,” *ACM Trans. Storage TOS*, vol. 7, no. 4, p. 14, 2012.
[60] R. W. M. Jones, “virt-mount.” [Online]. Available: http://libguestfs.org/virt-mount.1.html. [Accessed: 30-Jun-2014].
[61] “What Is CFEngine? - CFEngine.” [Online]. Available: https://cfengine.com/product/what-is-cfengine/. [Accessed: 03-Jul-2014].
[62] “What is Puppet? | Puppet Labs.” [Online]. Available: http://puppetlabs.com/puppet/what-is-puppet. [Accessed: 03-Jul-2014].
[63] “Chef | IT automation for speed and awesomeness | Chef.” [Online]. Available: http://www.getchef.com/chef/. [Accessed: 03-Jul-2014].
[64] S. Nelson-Smith, *Test-Driven Infrastructure with Chef: Bring Behavior-Driven Development to Infrastructure as Code*. O’Reilly Media, Inc., 2013.
[65] L. Kanies, “Puppet: Next-generation configuration management,” *USENIX Mag.*, vol. 31, no. 1, pp. 19–25, 2006.
[66] D. Lutterkort and M. McLoughlin, “Manageable virtual appliances,” in *Proceedings of the 2007 Ottawa Linux symposium*, 2007, pp. 293–302.
[67] S. Krum, W. Van Hevelingen, B. Kero, J. Turnbull, and J. McCune, “Developing and Deploying Puppet,” in *Pro Puppet*, Springer, 2013, pp. 73–96.
[68] “Vagrant.” [Online]. Available: http://www.vagrantup.com/. [Accessed: 03-Jul-2014].
[69] M. Hashimoto, *Vagrant: Up and Running*. O’Reilly Media, Inc., 2013.
[70] M. Hüttermann, “Infrastructure as Code,” in *DevOps for Developers*, Springer, 2012, pp. 135–156.
[71] J. Palat, “Introducing vagrant,” *Linux J.*, vol. 2012, no. 220, p. 2, 2012.
[72] M. Peacock, *Creating Development Environments with Vagrant*. Packt Publishing Ltd, 2013.
[73] “What Is Docker? An open platform for distributed apps.” [Online]. Available: http://www.docker.com/whatisdocker/. [Accessed: 03-Jul-2014].
[74] R. B.-A. Beslic, “Cloud Computing.”
[75] “How package management changed everything « Ian Murdock’s Weblog.” [Online]. Available: http://ianmurdock.com/solaris/how-package-management-changed-everything/. [Accessed: 03-Jul-2014].
[76] “virt-customize.” [Online]. Available: http://libguestfs.org/virt-customize.1.html. [Accessed: 03-Jul-2014].
[77] J. Cappos, S. M. Baker, J. Plichta, D. Nguyen, J. Hardies, M. Borgard, J. Johnston, and J. H. Hartman, “Stork: Package Management for Distributed VM Environments.,” in *LISA*, 2007, vol. 7, pp. 1–16.

[78] J. Kornblum, “Identifying almost identical files using context triggered piecewise hashing,” *Digit. Investig.*, vol. 3, pp. 91–97, 2006.
[79] G. H. Moulton and S. B. Whitehill, *Hash file system and method for use in a commonality factoring system*. Google Patents, 2004.
[80] M. McLoughlin, *The QCOW2 image format*. 2008.
[81] J. Kotek, “MapDB.” [Online]. Available: http://www.mapdb.org/. [Accessed: 01-Aug-2014].
[82] Dipartimento di Scienze dell’Informazione Università degli Studi di Milano, “The DSI Utilities.” [Online]. Available: http://dsiutils.di.unimi.it/. [Accessed: 01-Aug-2014].
[83] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm: the Linux virtual machine monitor,” in *Proceedings of the Linux Symposium*, 2007, vol. 1, pp. 225–230.
[84] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao, “Dissecting video server selection strategies in the youtube cdn,” in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, 2011, pp. 248–257.
[85] V. K. Adhikari, S. Jain, and Z.-L. Zhang, “Where do you‘ tube’? uncovering youtube server selection strategy,” in *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, 2011, pp. 1–6.
[86] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, “I tube, you tube, everybody tubes: analyzing the world’s largest user generated content video system,” in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, 2007, pp. 1–14.
[87] “W3Techs.” [Online]. Available: http://w3techs.com.
[88] “Debian Popularity Contest.” [Online]. Available: http://popcon.debian.org/. [Accessed: 30-Jun-2014].