Julius–Maximilians–Universität Würzburg
Institut für Informatik
Lehrstuhl für Kommunikationsnetze (Informatik III)

Performance Assessment of Service Migration Strategies in Cloud Environments

Leistungsbewertung von Migrationen von Internet-Diensten in Cloud-Umgebungen

Masterarbeit im Fach Informatik
vorgelegt von

Lukas Iffländer

Julius–Maximilians–Universität Würzburg
Institut für Informatik
Lehrstuhl für Kommunikationsnetze (Informatik III)

Performance Assessment of Service Migration Strategies in Cloud Environments

Leistungsbewertung von Migrationen von Internet-Diensten in Cloud-Umgebungen

Masterarbeit im Fach Informatik
vorgelegt von

Lukas Iffländer
geboren am 26. April 1990 in Neustadt a. d. Waldnaab

Angefertigt am
Lehrstuhl für Kommunikationsnetze (Informatik III)
Julius–Maximilians–Universität Würzburg

Betreuer:
Prof. Dr.-Ing. P. Tran-Gia
Dr. rer. nat. Florian Wamser

Abgabe der Arbeit:
19.11.2015

Erklärung

Ich versichere, die vorliegende Masterarbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur verfasst zu haben. Darüber hinaus versichere ich, die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde zur Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 19.11.2015

(Lukas Iffländer)

Deutsche Zusammenfassung

Der Begriff *Cloud* ist derzeit in aller Munde. Cloud-Dienste werden jeden Tag beliebter. Jedes Smartphone benutzt die Cloud. Jedes Tablet und jeder Internet-fähige Fernseher greifen darauf zurück. In gleicher Weise werden beim Arbeiten im Büro der Zukunft, anstelle von Offline-Anwendungen, Online-Programme aus der Cloud aufgerufen, die auf jedem Internet-fähigen Gerät weltweit nutzbar sind. Dadurch entwickelt sich die Cloud für die Hosting-Anbieter zu einem immer größeren Geschäft. Amazon AWS beispielsweise erreicht inzwischen einen Umsatz von 1.57 Milliarden Dollar pro Quartal, als einer der beliebtesten Anbieter. Der Dienst erwirtschaftete in der selben Zeit einen Gewinn von 264 Millionen Dollar [1]. Die zugrunde liegende Infrastruktur umfasst inzwischen zwei Millionen Server, die von über einer Million Nutzer verwendet werden.

In dieser Arbeit konzentrieren wir uns auf persönliche Cloud-Dienste wie personalisiertes Video-Streaming, Online-Gaming oder Dienste zur netzgebundenen Datenspeicherung. Ein Cloud-Dienst ist dabei ein virtuelles PC-Image, das beliebig in Datenzentren oder Cloud-Hostern instanziiert werden kann. Insbesondere kann es dynamisch in einem Datenzentrum und über Datenzentrumsgrenzen hinweg verschoben und platziert werden.

Cloud-Dienste bieten den Nutzern nahezu unendlich viele Rechenressourcen, die im Bedarfsfall flexibel ausgelastet werden können. Besonders für Anwendungen mit unausgewogenem Lastprofil ist dies eine sehr attraktive Option, da die Ressourcen nur gebucht und bezahlt werden müssen, wenn sie benötigt werden. Diese Dienste können je nach verfügbaren Netzwerkressourcen, Anforderungen des Dienstes, Qualitätserwartungen des Kunden und Kosten optimiert werden. Optimieren heißt in diesem Fall den Dienst zu duplizieren, zu skalieren oder zu platzieren. Man nennt diesen Prozess allgemein *Ressourcenmanagement* für Cloud-Dienste. Der Prozess des Verschiebens von Cloud-Images innerhalb oder zwischen Rechenzentren nennt man *Migration*.

Wenn auch die technischen Voraussetzungen für Migrationen in aktuellen Systemen gegeben sind, so bestehen doch weiterhin offenen Fragestellungen:

1. Unter welchen Netzwerk- und Umgebungsbedingungen ist es möglich einen Dienst zu migrieren?
2. Was ist der Einfluss einer Migration auf den Dienst?
3. Wie wirken sich Netzwerkparameter und Netzwerkeinstellungen auf die Migrationsdauer und die Dienstqualität aus?
4. Welche Informationen werden gebraucht, um eine Migration zuverlässig durchzuführen?
5. Wie kann man die Leistung einer Migration messen?

Greift ein Nutzer beispielsweise auf einen Cloud-Dienst zu, der in einem Rechenzentrum lokalisiert ist und erleidet die Netzwerkverbindung zwischen dem Nutzer und dem Rechenzentrum einen signifikanten Qualitätseinbruch, so muss durch das Ressourcenmanagement entschieden werden, ob eine Migration in ein anderes Rechenzentrum durchgeführt werden soll. Hierfür muss unter anderem abgewägt werden, ob die bessere Dienst-Qualität den Migrationsaufwand rechtfertigt.

Mehrere Studien haben sich bisher mit den Thema Virtualisierung in Cloud-Umgebungen beschäftigt. Ein Teilgebiet behandelt dabei die Platzierung von virtuellen Maschinen innerhalb der Cloud-Infrastruktur, was die Grundlage für Migrationen darstellt [2, 3, 4, 5]. Andere Studien behandeln die Realisierung von Migrationen zwischen unterschiedlichen Cloud-Umgebungen [6] oder fokusieren sich auf die Modellierung und Leistungsbewertung von Virtualisierungen beim Einsatz in Cloud-Umgebungen [7, 8, 9, 10, 11].

In dieser Masterarbeit werden Dienst-Migrationen definiert, implementiert und evaluiert. Die Arbeit leistet den folgenden Beitrag zu den zuvor genannten Themen und Problemstellungen:

- Literaturrecherche zum Thema: Cloud-Dienste, Ressourcenallokation, Orchestration, Migration.
- Strukturierung eines Testbeds mit Cloud-Datenzentrum, Dienst-PCs, Netzwerkemulatoren zur Emulation verschiedener Netzwerkbedingungen zwischen den Komponenten.
- Erstellung eines Mess-Frameworks zum automatischen und wiederholten Durchführen von Experimenten im Testbed.
- Bewertung und Messung von verschiedenen Migrationsarten im Testbed.
- Bewertung der Leistung und Quantifizierung des Nutzens der Migration im Hinblick auf den Nutzer (Quality of Experience, Applikationsparameter) und das Netzwerk (Ressourcenausnutzung, Kosten).

Contents

1 Introduction

Nowadays everyone talks about the *Cloud*. Cloud services are becoming more and more popular every day. Every smartphone uses the cloud. Every tablet and every Internet capable television set is accessing it. In a similar fashion offline applications are replaced by online services in the office of tomorrow. Such an office is accessible everywhere and usable on every Internet capable device world-wide. This trend translates to a lucrative business for the cloud hosting providers. Amazon AWS, for example, has reached a turnover of US$1.57 billion as one of the most popular providers. At the same time, the service generated a revenue of US$265 million [1]. The underlying infrastructure has grown to two million servers with over one million customers.

In this work, we focus on personal cloud services like personalized video streaming, online gaming and network storage services. A cloud service is a virtualized PC image that can be arbitrarily instantiated in data centers or cloud platforms. Especially, it is able to be placed and moved across data center borders.

Cloud services offer almost endless resources to the user, which can be flexibly allocated when necessary. This is a very attractive option particularly for applications that feature an unbalanced load profile since the users only need to book and pay for resources when they are needed. This service allocation can be optimized according to available network resources, application requirements, the customers' quality expectations or cost. Optimizing in this case means duplicating, scaling or placing the service. This approach is called *resource management* for cloud services. The process of moving a cloud service inside or between data centers is called *migration*.

Even though, the technical preconditions for migrations in current systems are fulfilled, multiple open questions remain:

1. Under which network and environment conditions is it possible to migrate a service?
2. What influence does the migration have on the performance of the migrated service?
3. How do network parameters and settings influence the migration duration and the service quality?
4. What information is required to reliably perform a migration?
5. How can the performance of a migration be measured?

Imagine, for example, a user that is accessing a cloud service in a data center. Now the link between data center and user suffers a severe loss of quality. Then the resource management must decide if a migration to another data center should be performed. For this decision it is necessary to decide whether the increased service quality justifies the migration effort.

Multiple studies have engaged in the field of virtualization in cloud environments. A particular research area covers the placement of virtual machines inside the cloud infrastructure that is the basis for migrations [2, 3, 4, 5]. Other studies cover the realization of migrations between different cloud environments [6] or focus on the modelling and performance evaluation of virtualization when used in cloud environments [7, 8, 9, 10, 11].

In this master thesis service migrations are defined, implemented and evaluated. The work has the following contribution to the aforementioned topics and problem definitions:

- Literature research on the topics of cloud services, resource allocation, orchestration and migration.
- Structuring of a testbed with cloud data center, service hosts, network emulators to emulated different network conditions between the components.
- Creation of a measurement framework for the automatic and repeated execution of experiments in the testbed.
- Evaluation and measurement of different types of migration in the testbed.
- Evaluation of the performance and quantification of the benefit of the migration in regard to the user (quality of experience, application parameters) and the network (resource utilization, cost).

The remainder is structured as follows. After this introduction in Chapter 1, an introduction to the technical background follows in Chapter 2. Afterwards, the related work is summarized in Chapter 3, and in Chapter 4 the cloud solution open stack is described. Then, the used testbed is defined in Chapter 5. Chapter 6 focuses on the evaluation of service migration strategies in cloud environments. Finally in Chapter 7, a conclusion is drawn and the work is summarized. In the Appendix some tables can be found facilitating the understanding of some of the used figures.

2 Technical Background

This section provides the essential background information on the technologies and concepts used in the thesis. It serves as a basic summary for the future understanding of this work. It starts with a basic definition of cloud infrastructure solutions. Afterwards, different types of cloud service migrations are presented. Then, the description of quality of experience management for applications is introduced. Next, representational state transfer is explained followed by the concept of a scalable logging structure.

2.1 Cloud Infrastructure Solutions

A cloud infrastructure consists of multiple components to enable dynamic and flexible management of the provided services. These components can be deployed independently on the same node as other modules as well as on dedicated nodes. For redundancy and scalability reasons some of these components can be set up on multiple machines. Typical cloud infrastructure solutions are OpenNebula, Apache CloudStack and OpenStack.

2.1.1 Classification and Description

Cloud infrastructures like Amazon AWS consists of the following entities:

Virtual Machines are an emulation of a particular computer system. From the perspective of the user, they function like normal non-virtual machines.

Hosts are the nodes are rung on the cloud services and especially where the virtual machines managed by the cloud environment are located. Usually those are hardware nodes but the hosts can also be virtual machines. Even cloud in a cloud solutions are possible.

Hypervisors are a piece of software, hardware or firmware that is capable of creating and running machines. A computer running a hypervisor is called a host machine.

Services are different applications offered directly to the end user including computation resources, databases or storage. The services usually are the base for the billing process for the cloud usage where the common mode of billing is utilization times duration of that utilization.

Virtualization hypervisors can be classified as

Type-1 specifying bare-metal or native hypervisors. They run directly on the host's hardware to control the hardware as well as manage the guest operating systems. These run as a process on the host node. Common examples are Oracle VM Server, Citrix XenServer, VMware ESX/ESXi and Microsoft Hyper-V 2008/2012.

Type-2 specifying hosted hypervisors. They run inside a conventional operating system like any other program. They abstract the guest operating system from the host operating system. VMware Workstation, VMware Player, VirtualBox and QEMU are examples of type-2 hypervisors.

The distinction is not entirely clear since there are application like Linux's Kernel-based Virtual Machine (KVM) and FreeBSD's bhyve are kernel modules effectively converting the host operating system into a type-1 hypervisor while at the same time competing for resources with any other non virtualization process running on the system which is more typical for type-2 hypervisors.

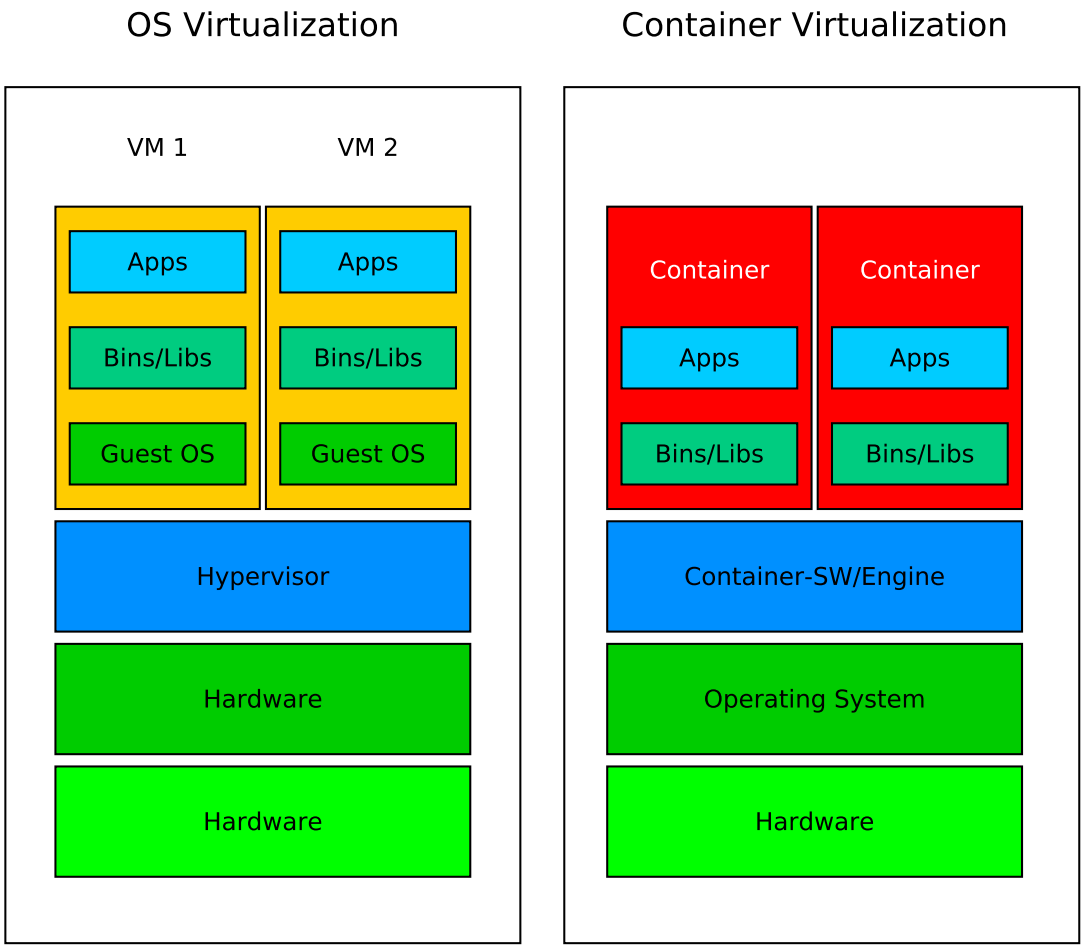

Figure 2.1: Modes for hosted virtualization

For the hosted virtualization there are multiple ways of realization as depicted in Figure 2.1. One option is the operating system virtualization completely virtualizing the guest operating system. Thus a full operating system is installed inside the virtual machine. This has the advantage that a total separation between the host and the guest system is achieved. The other option is to just provide so called software containers. They provided a process with the before specified required libraries but the process itself runs on the host operating system in an isolated user-space instance. Common implementations of container virtualization are Linux Container (LXC), OpenVZ, Docker and VMware ThinApp.

2.1.2 Existing Cloud Infrastructure Solutions

OpenNebula

OpenNebula is a cloud computing platform for managing heterogeneous distributed data center infrastructures. It is designed to build private, public and hybrid implementations of infrastructure as a service. It is sponsored by OpenNebula Systems. It considers itself filly enterprise ready. It focuses more on the task of data center virtualization than infrastructure provision and is very flexible. Support is directly provided by the developer team.

Apache CloudStack

Apache CloudStack is an open source cloud computing software for creating, managing, and deploying infrastructure cloud services. It uses existing hypervisors such as KVM, VMware vSphere, and XenServer/XCP for virtualization. It was originally developed by Cloud.com and was released as free software in 2010 with about five percent of the code base kept proprietary. In 2011 Cloud.com was purchased by Citrix who released the remaining code. In 2012 the development finally switched to the Apache Software Foundation (ASF). CloudStack is considered enterprise ready. Compared to OpenNebula CloudStack offers less flexibility and shifts the focus slightly more to infrastructure provision while still focusing on data center virtualization. CloudStack offers interfaces for many of the OpenStack services allowing mixed cloud environments. Direct support is offered from the Apache Foundation.

Eucalyptus

Eucalyptus is free and open-source computer software for building Amazon Web Services (AWS)-compatible private and hybrid cloud computing environments marketed by the company Eucalyptus Systems. Eucalyptus stands for Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems. It currently is owned by Hewlett-Packard. Eucalyptus provides a simpler configuration than the competing solution but at the same time suffers from a severe loss of flexibility. The system in generally considered enterprise ready. It focuses complete on the infrastructure provision. Direct support from the developer team is available.

OpenStack

OpenStack is an open source platform for cloud computing. The project was initiated by Rackspace Hosting and NASA. It is widely spread and offered by dozens of hosting providers. Due to its massive number of modules OpenStack covers the area of data center virtualization as well as the sector of infrastructure provision. The high flexibility comes at the price of the massive complexity resulting in a massive setup effort for the cloud environment. There are tools like dropstack or devstack trying to assist in the creation of the environment. They usually come at the disadvantage of either reducing the flexibility or are almost as complicated to setup than the OpenStack environment itself. The complex setup makes OpenStack prone to configuration errors leading to some hosters not considering it enterprise ready. Another problem is, that no direct professional support from the developers is available but instead provided by the providers that offer OpenStack based solutions. Of the listed cloud solutions OpenStack is the most heavily developed. This means on one hand that new features and technologies are quickly implemented but also that the incompatibilities between major releases often require countless hours to migrate or even a complete re-installation.

2.2 Cloud Service Migration Approaches

2.2.1 Migration Types

In cloud environments it is often necessary to move services from one host or even one compute center to another. There are three different migration types:

Data Migration: Only the local data is moved to another host. The application is already running on this host and continues operation with the migrated data. This is the migration mode requiring the least amount of data but also the most intelligence in the application to move the data. An exception are stateless applications where switching between two hosts that are running the same application requires no data migration.

Software-/Application Migration: The complete software is moved to the target host. The target and the source keep their operating system as well as applications that do not depend on the migrated application running without modifications. For state-full migrations this migration usually includes a data migration.

Hardware migration: In case of the hardware migration the complete compute unit is moved from one location to another. In the common scenarios that means shutting down the machine, removing its hardware connections, transport it to the target location, install the hardware and finally resume the machine operation. Since the software and the data are left on the drive this includes an application migration as well as a data migration. A special variant, commonly used in cloud infrastructures is the migration of virtual machines. Since a virtual machine simulates real hardware the movement of a machine from one

host to another can be described at virtualized hardware migration. It usually requires the same steps as the regular hardware exception but the attached connections are virtually disconnected and reconnected and the transport is not done by hand but via the network.

2.2.2 Migration Modes for Virtual Hardware Migration

Different approaches exist to the virtual hardware migration. They require different levels of software complexity and offer different levels of service quality regarding migration duration and downtime.

Traditional Approach

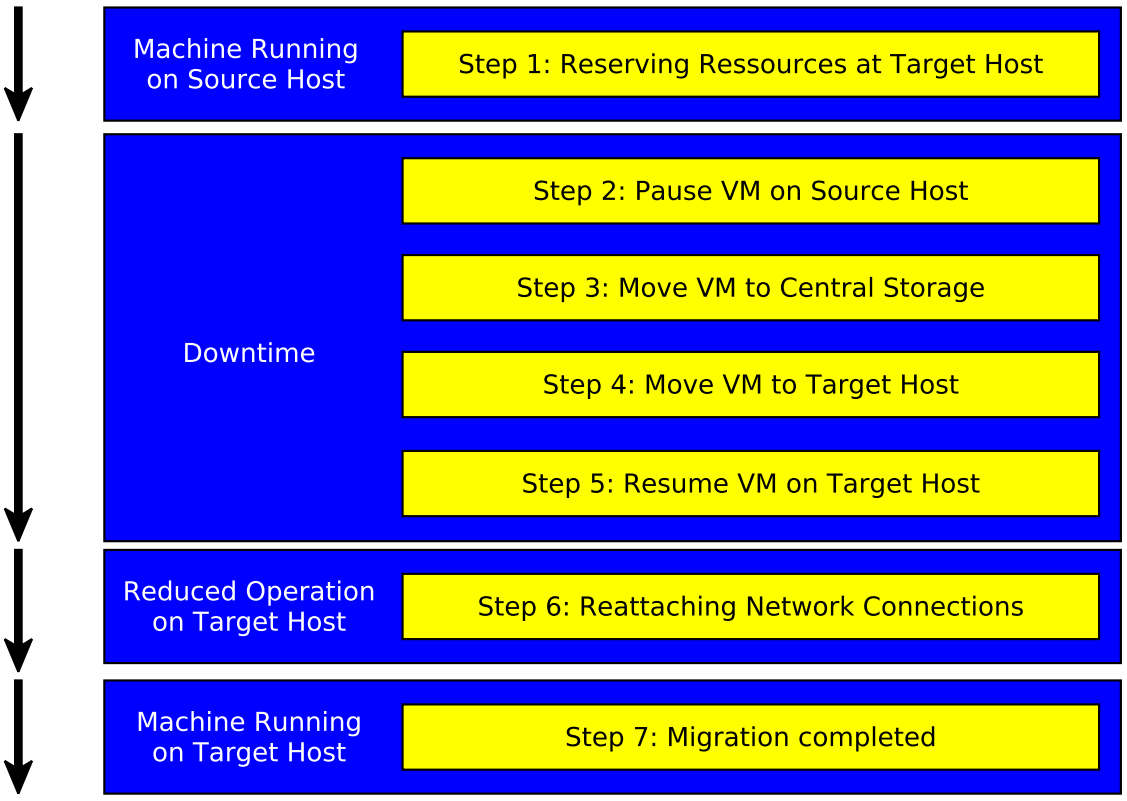

Figure 2.2: Traditional Migration Approach

The classical approach as shown in Figure 2.2 follows a process similar to the migration approach for physical hardware. After the migration is requested the system checks whether the target host has enough resources (e.g. virtual CPUs, Memory, Storage, ...) available. If this is the case these resources are reserved.

Then the machine on the source host is then paused. The snapshot of the paused machine is then moved to a central storage node. Afterwards it is moved to the target host, where it is resumed. After the system is resumed it continues the operation but has not yet access to the network which is attached in the next step. After this the migration is complete.

The machine is down for the complete time it is transferred resulting in a very long downtime depending on the network speed and link quality. The advantage of this mode is that it is very simple. In optimized variations of this mode the transfer of the snapshot image is not done over central storage but instead directly between the two hosts in the best case cutting the transmission time in half. Still the migration can take very long and most services will not survive a connection loss this long without operation failures.

Pre-Copy Migration

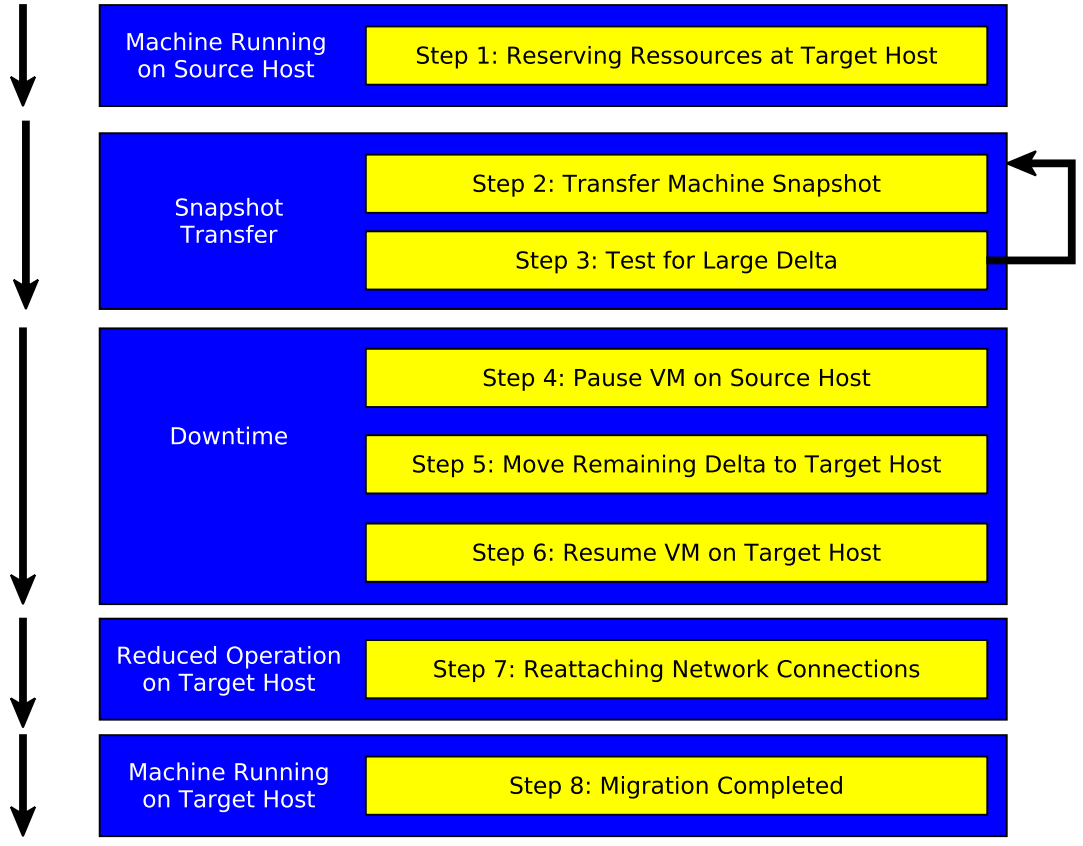

Figure 2.3: Pre-Copy Migration

The Pre-Copy-Migration as shown in Figure 2.3 focuses on getting a fully working machine across the network while reducing the downtime. Therefore after the reservation of the resources on the target host a snapshot of the virtual machine is created and transferred while the machine is still operating on the source host. Since the machine is continuing to perform workloads the system after the transmission of the snapshot the hypervisor checks how large the accumulated delta is is. If the delta is to high a new delta snapshot is transmitted. This process is repeated until the delta falls under a preset threshold. When this is the case the machine is paused. The remaining delta is transferred and the machine is resumed on the target host. Finally the network connection is reattached.

The downtime of the machine is kept very short since it is only paused while the remaining delta (if at all existing) is transfer ed. Therefore the duration of the downtime depends of the preset value for the threshold. One disadvantage is the recursive progress of sending delta snapshots creating additional network transfers compared to the classical approach. Selecting the threshold is complicated. If it is set to high the downtime takes to long and services fail. If set to low the delta might be constantly to high and the migration never happens. Since this migration type tries to do the migration en block it is also called the block migration.

Post-Copy Migration

Figure 2.4 depicts the process of the Post-Copy-Migration. The goal of the Post-Copy-Migration is to get the virtual machine working on the target node as fast as possible. Therefore the machine is stopped right after the resource reservation. Then a minimal memory image is moved to the target node. It contains CPU state, registers and optionally non-pageable memory. The VM is then resumed based on this minimal subset of the execution state and the network is attached to the machine on the target node. Concurrently the remainder of the complete machine image is transferred over the network. If the VM at the target tries to access memory not yet transferred it creates a page fault, also known as network fault. The network faults are trapped and redirected to the source node which responds by sending the faulted page. The migration is completed after the complete remainder of the image is transferred.

The downtime is kept quite short and the goal of moving an operational machine to the target as fast as possible is achieved. The quality of the operation is dependant on the number of network faults. If the number of faults is to high, the application running on the node can be massively impaired or even fail. Therefore this approach requires a lot of intelligence in the migration mechanism predicting which parts of the memory will be required and migrating them first as well as deducing access patterns from previous faults and migrating memory located around the faulty blocks.

Live Migration via Central Storage

The last approach is the so called Live-Migration. The process is visualized in Figure 2.5. Different to the aforementioned concepts the live migration does not

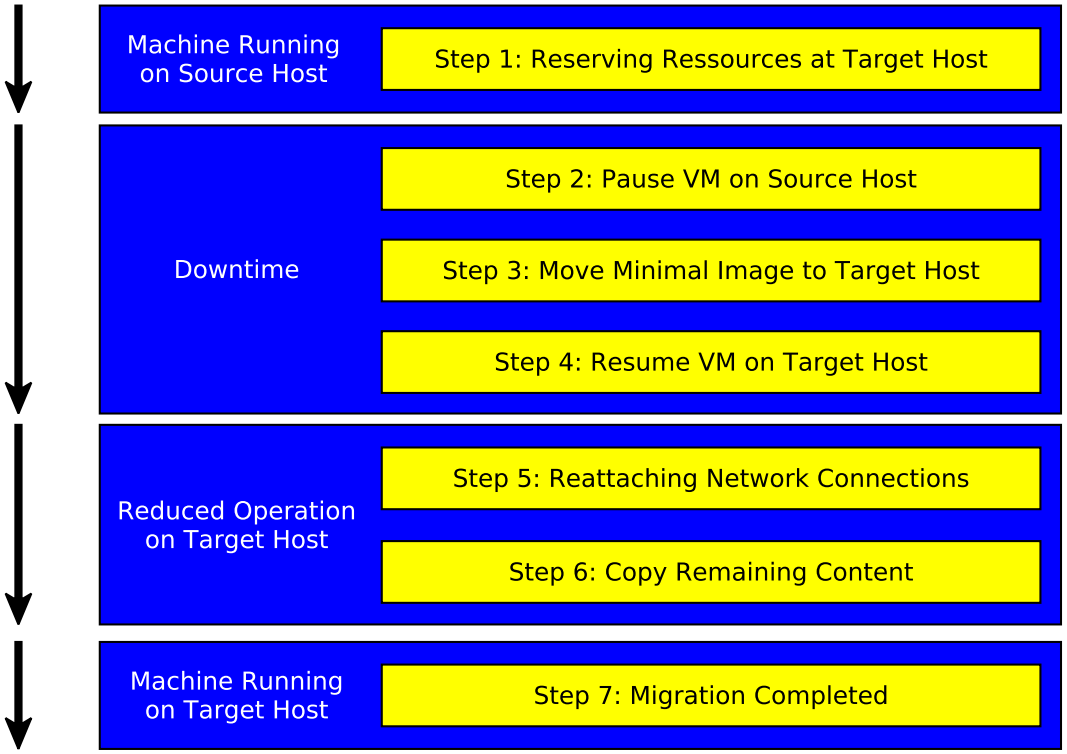

Figure 2.4: Post-Copy Migration

store the instances of the virtual machines running in the cloud environment on the compute nodes themselves but instead on a central storage. After the reservation of the target's resources the VM is paused on the source node. Now, if caching of the network storage was utilized, the network storage is synchronized at first for the source node and at second for the target node. Then the machine is resumed on the target node and the network is attached. If the system runs on a very fast network ($\geq$10Gbps) without cache the synchronization steps can be skipped.

The live migration is very fast and minimizes the downtime. Even though it has its disadvantages. The usage of central storage increases the required network bandwidth during normal operation while at the other hand reducing the transfer required during the migration. Also the connection to the storage and the storage it self can be a bottle neck since the network link is usually slower than locally attached SAS drives and multiple VMs accessing the central storage can lead to the central storage's drives being used at full transmission capacity while still not being able to handle all requests. Also the chosen type of network storage can influence the performance. Either classical network file systems like NFS on CIFS can be used or block storage services provided by the employed cloud environment.

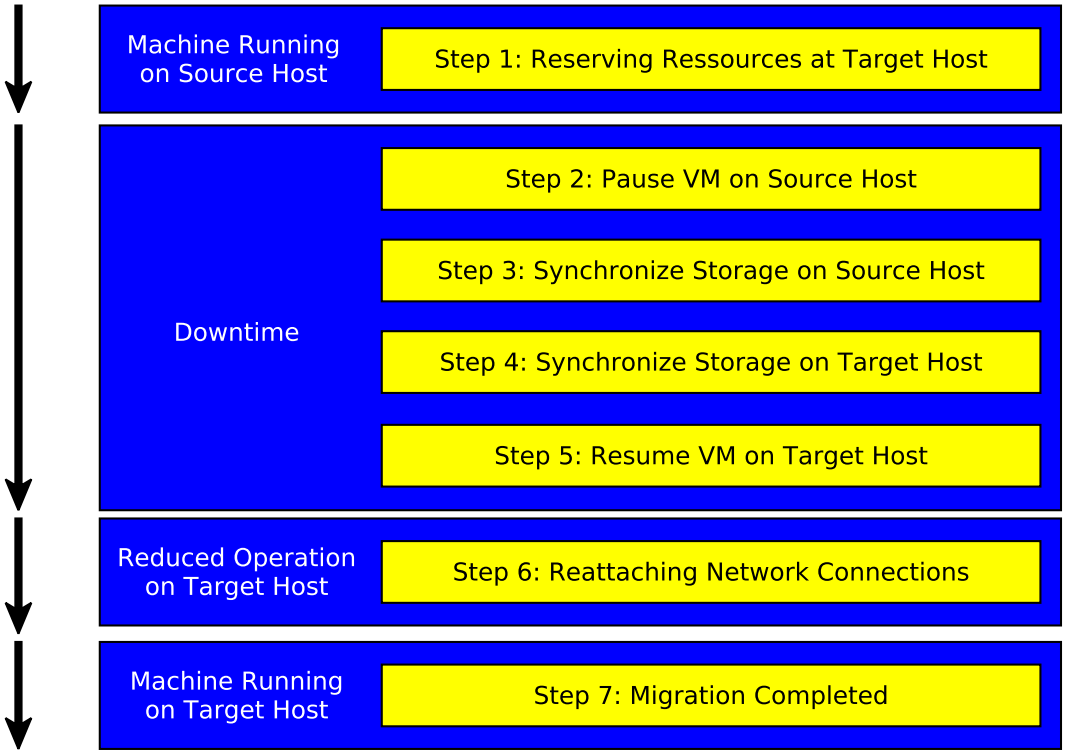

Figure 2.5: Live Migration

2.3 Quality of Experience Management for Internet Applications

A central topic in today's research in the field of network technologies is the Quality of Experience (QoE). The overall goal of many publications is to offer ideas to increase that factor in certain situation.

2.3.1 Definition

"Quality of Experience (QoE) is the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and/or enjoyment of the application or service in the light of the user's personality and current state.[12]"

In the hourglass model the quality of experience provides the topmost layer as show in Figure 2.6. Mapped to the IP hourglass model this would be the user layer. It is visible from the figure that QoE is dependant on the quality levels of the lower

layers. It depends on the Quality of Transmission (QoT), Quality of Service (QoS), Quality of Delivery (QoD) and Quality of Presentation (QoP).

Quality of Transmission (QoT) is the bit error rate of the physical network link.

Quality of Service (QoS) "is the idea that transmission rates, error rates, and other characteristics can be measured, improved, and, to some extent, guaranteed in advance[13]." In the modern Internet QoS is mainly done at the IP level.

Quality of Delivery (QoD) is the concept that the selection and or modification of the right transmission protocol and the handover between the protocol and the application has a measurable influence on the users experience.

Quality of Presentation (QoP) is the quality the arrives at the medium the user uses to consume the delivered content. It depends on the fact that the user is presented with all the request data and in which way the data is displayed.

QoE is one of the most important metrics for resource management of end user applications[14]. Utilizing this metric requires three steps: QoE modeling, monitoring and optimizing[15].

2.3.2 Modeling

Integrating the QoE into the resource management requires a fundamental understanding of the varying application requirements as well as the impairment of the quality perceived by the user from network disturbances. Therefore the QoE needs to be modeled for every specific application with different network parameters[15].

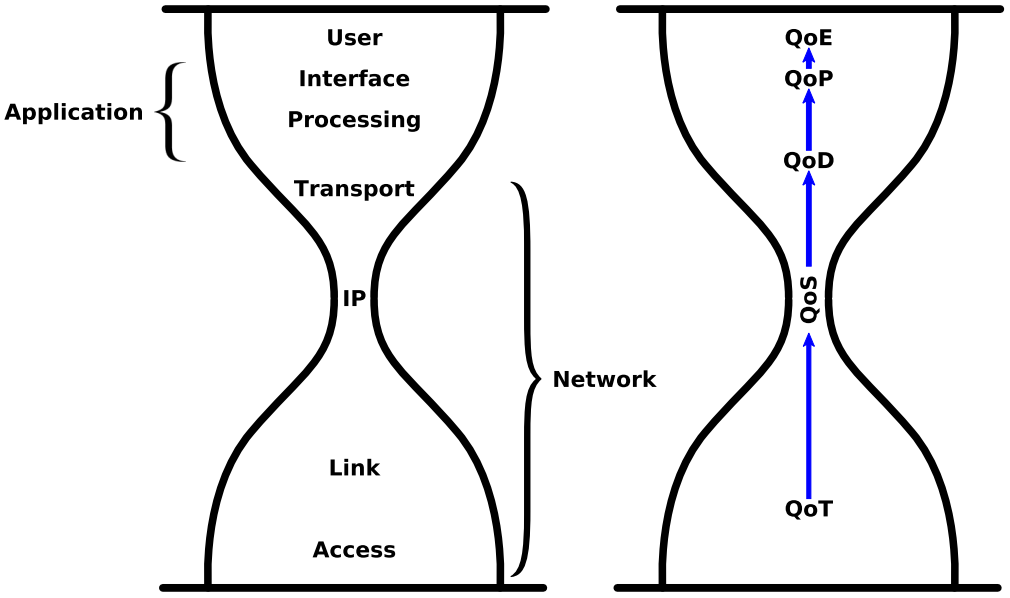

Figure 2.6: Hourglass models of IP networks besides the model for the different quality layers.

The modelling consists of two parts. One is quantifying the perceived quality by the user and the second is measurement of application related metrics depending on the network connection quality[14]. The second part can be turned into a model for application layer QoS. Based on this a mapping to the user perceived quality has to be done by subjective user studies.

2.3.3 Monitoring

Resulting from the modeling the parameters relevant to QoE are identified. To monitor the QoE based on the model created before these parameters need to be monitored. Usually the parameters include "(i) the network environment (e.g., fixed or wireless); (ii) the network conditions (e.g., available bandwidth, packet loss[, latency]); (ii) terminal capabilities (e.g., CPU power, display resolution); (iv) service and application-specific information (e.g. video bitrate, encoding, content genre)[15]."

Whilst direct QoE monitoring has the advantage of being very precise it requires very complex models and therefore creates either more computational load or makes near real-time monitoring impossible when crowd sourcing is used for the QoE quantification[16].

Monitoring the application state on the other hand in some cases may not be as precise as QoE monitoring but the complexity is vastly reduced while keeping which also reduces the level of computational load. In many cases the used QoE metric is also an application state. In these cases no precision is lost.[17].

When monitoring only the application state it is necessary to use a state that is either a QoE metric itself or is deductible to a QoE metric. Therefore, the application state is directly correlated to the users' QoE. The monitoring of this state can be done by probing network parameters as well as directly getting application information.

The monitoring can be done at different locations. At the client it can be done by modifying existing applications or appending an additional monitoring application to the monitored one. Since the information is directly derived from the application this approach is very precise and the additional processing requirements are negligible.

Distributed monitoring (e.g. at a Gateway) is still precise but requires a higher amount of computing power. The slow hardware deployed in most internet gateway devices and the remote detection of user interaction with the application increase the level of complexity since the used algorithms must make do with the available hardware and try to predict interactions such as canceling or pausing a video on the client machine[18].

Moving the monitoring to a centralized point in the network (e.g. at the provider) increases the the amount of data that has to be monitored requiring very high processing power. It inherits the interaction problem from the distributed monitoring approach. In the following concepts to tackle these problems are presented.

For reference a short comparison of the different location approaches is shown in Table 2.1.

	Precision	**Processing Requirement**	**Complexity**
Client	accurate	low	application modification
Distributed (e.g. Gateway)	good	high	slow hardware user interaction
Centralized	good	very high	huge amount of data user interaction

Table 2.1: Comparison Application Monitoring Locations

2.3.4 Optimizing

After the modeling and the monitoring the final step is to use the learned information from the model to optimize the QoE in the resource management with the goal of minimizing the dissatisfaction upon the user base. The QoE control targets to take action before the QoE declines. The QoE resource management addresses the questions, "(a) where to react, i.e. at the edge, within the network or both; (b) when to react and how often; and (c) how to react and where which control knobs to adjust[15]".

2.4 Representational State Transfer

The term REpresentational State Transfer (REST) was introduced by Roy Fielding [19] in the year 2000. REST is defined as a network oriented architectural style, with the objective to minimize the latency as well as the communication overhead over the network and at the same time increase the components independence and scalability.

Rest follows many different design principles. Goals are the support of caching, the dynamical sustainability of components and the processing of actions at intermediate nodes (e.g. gateways or load-balancers). This features are designed to enable high scalability, expansion and success of the World Wide Web that is an interface of the REST architecture style. Nevertheless REST is not tied to HTTP as a protocol and only describes the abstract properties required for a REST-compliant system but HTTP is the most commonly protocol used with REST APIs. REST corresponds to the need of the Internet Engineering Task Force (IETF) for a behavioral model of the Web. The format of the data transmission can vary too. While JSON is the most common format there are also implementations using XML or CSV files.

CRUD	REST	Description
CREATE	PUT/POST	Create and set the state of a resource
READ	GET	Retrieve a resource's current state
UPDATE	PUT/PATCH	Modifies a resource state
DELETE	DELETE	Removes a resource

Table 2.2: CRUD operations as implemented in REST

REST implements all CRUD (Create, Read, Update, Delete) operations necessary for the interaction with other services as shown in Table 2.2. REST is view as a simpler alternative to the competing SOAP and WSDL[20] based services.

An important characteristic of the REST approach is that it is stateless in the client server interaction. No client context is stored at the server. Therefore a successful request must contain all the required information needed for the processing and if required any session state is handled by the client. This facilitates server side scalability and the introduction of failure tolerance but increases the network load due to the re-submission of already transferred information.

2.5 Logging Infrastructure

The Logstash book[21] describes log management with the following fitting words. "Log management is often considered both a painful exercise and a dark art." The understanding of a good log management is a slow and evolutionary process. Many system administrators use tools like `cat`, `tail` or `grep` but these do not scale for large scale environments with multiple servers. Analyzing log files on hundreds or even thousands of servers to find the source of a failure is simply not feasible.

Addressing of this problem requires a centralized infrastructure concentrating the log files. But still there is the problem of too much information and too little context as well as different log formats (different time-stamps etc.) that make it hard to find what one is looking for with the aforementioned tools.

Therefore the log management must be able to filter and parse the received log files into a common format that is easily searchable by a searching and indexing technology while at the same time being able to scale with growing network size and number of log files processed.

The log management can be divided into several modules:

Shipper The shipper runs at the node where the log files are created. It collects the log files and if necessary it can perform preprocessing (e.g. drop certain log entries since they are not needed and would only create additional network traffic) and then forwards them to the broker.

Broker the broker handles the transmission between the shipper nodes and the indexer nodes (e.g. by providing a message queue).

Indexer the indexer processes the received log files converting them in the target format while indexing the important information. The indexer then adds the parsed information to the search cluster.

Search Cluster the search cluster consists of one or multiple nodes providing the ability to sort and search the data entered by the indexer.

All these nodes on the way can be scaled to support scaling of the network or to introduce fault tolerance. An example for this is shown in Figure 2.7. From each shipper there are multiple ways of getting the information from the logs to the

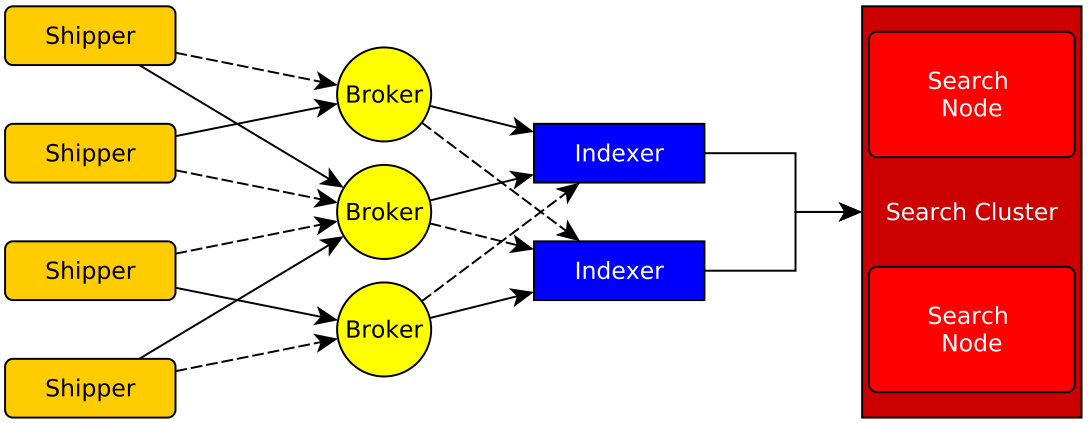

Figure 2.7: Architecture of a scalable logging infrastructure. Dashed lines mark backup links.

search cluster. This allows parallel processing of multiple logging sources as well as continuous operation if one of the nodes on the way should fail.

3 Related Work

Many studies have been performed in neighbouring topics occasionally touching the topic of the thesis. In the following a sample of these are described and their results are summed up.

3.1 Selection of Migration Destination

In [2] "*A Network-aware Virtual Machine Placement and Migration Approach in Cloud Computing*" is proposed. Since virtual machines in cloud infrastructures are solely stand-alone applications they need to interact with each other over the network. Therefore the network I/O performance often is a key factor to the application performance. Therefore the authors suggest in their approach a placement of the machines to minimize the data transfer time consumption. They validate their approach by simulation. The utilized approach is utilizes a data distribution matrix accounting for the status of the data distribution of the applications. Based on this matrix the distribution is minimized with regard to the available resources at certain host nodes. The validation shows that indeed a gain is achieved with the duration of certain network heavy tasks reduced up to 25%.

In [3] "*A Novel Virtual Machine Placement in Cloud Computing*" is presented. Common to [2] the key problem definition is the minimization of the amount of data transferred between different machines in the cloud environment. They use a similar approach as before using matrices to characterize network parameters and available resources. Unlike in [2] a larger number of characteristics is used and in a more formal approach the optimization formula that can be fed into linear programming is established. The authors also use simulation for their validation having similar results to the previous source.

The paper in [4] discusses "*Autonomic Virtual Machine Placement in the Data Center*" presenting a high level overview of a virtual machine placement system. A controller is designed to map virtual machines to the host according to user-specified policies. By monitoring the machine activity and employing advanced policies the authors claim to be able to achieve substantial cost savings from better utilization of computing resources and less frequent overload situations. The evaluation was done by experiment as well as simulation. While at the beginning of the validation setup the amount of imbalance exceeds the selected threshold by migrating the load is balanced with equal load over all hosts for CPU, LAN as well as SAN. Where the SAN load was balanced first and the other twos slightly later.

In [5] an approach to "*Migration-aware Optimization of Virtualized Computational Resources Allocation in Complex Systems* is presented. The novelty is the migration of application assigned to the resources of one machine can be migrated

to another during system lifetime. This is done by using the proposed heuristic optimization. Since the valid solutions are non-convex the procedure is decomposed in a relax-and-round approach. The proposed decomposition facilitates fast algorithm convergence while still guaranteeing the solution satisfies the assumed constraints. The evaluation for multiple environments with between three and 80 nodes shows that the approximation is near to the optimal solution.

3.2 Migration Between Different Cloud Platforms

In an the article [6] the topic of "*Application Migration Effort in the Cloud - The Case of Cloud Platforms*" is discussed. They specify that a problem is the migration requiring development effort and open up new risks of vendor lock-in. They present a cloud-to-cloud migration between seven different cloud platforms to make applications more independent of the used cloud provider and the employed cloud management software. They use a Docker-based deployment system enabling the comparison of different platforms for particular applications. The approach is validated by implementing it for the used platforms. The migration between vendors requires trade-offs hand changes to the technology setup. With the developed tool using container virtualization and migration it was possible to deploy onto different cloud systems without making these trade-offs. The results show that major differences between the cloud providers exist. VM-based platforms require more effort than container based platforms. The results show that platform independent development is a very complex topic taking a lot of resources.

3.3 Modelling of Virtualized System Performance

In the dissertation of Nikolaus Huber [7] introduces modeling technologies for for "*Autonomic Performance-Aware Resource Management in Dynamic IT Service Infrastructures*". He argues for the need of sophisticated modeling technologies for static and dynamic aspects of the managed system as well as the requirement to be able to adapt these technologies at run time. He presents core concepts like a process model for proactive model-based system adaption using the Descartes Modeling Language (DML). He introduces modeling abstraction to describe complex IT infrastructures like cloud environments as well as the degrees of freedom in such a system and presents a method for the identification, classification, and automatic quantification of performance-influencing properties. He specifies VM migrations as one approach to the dynamic adaption of the resource management.

In [8] an "*Analysis of the Performance-Influencing Factors of Virtualization Platforms*" is presented with the goal of establishing a generic approach to predict the performance influences on virtualization platforms. Afterwards, a general methodology to quantify the influence of the identified factors is presented which is demonstrated by a case study.

The paper in [9] evaluates and models virtualization performance overhead for cloud environments. The problem is that virtualization and the sharing of resources have direct effects on application performance making performance prediction of

cloud deployed services very complex. The authors propose a generic performance prediction model for the two different hypervisor types.

In [10] "*Automated Modeling of I/O Performance and Interference Effects in Virtualized Storage Systems*" is discussed. Storage in virtualization environments can quickly become a bottle neck. This topic has so far been avoided due to the complexity of modern virtualized storage systems leading to the common deployment of solution sized by guessing. The paper presents an automatic approach to modelling this I/O performance. The model is validated in three case studies on real-world hardware. The results are very precise with a mean prediction error of up to 7%.

The paper in [11] discusses"*Predictive Performance Modeling of Virtualized Storage Systems using Optimized Statistical Regression Techniques*" presenting another approach at I/O modeling. This approach uses a general heuristic search algorithm optimizing the parameters of regression techniques. The approach is then validated in a in-depth evaluation in a real-word environment. The optimizations are capable of reducing the error from [10] by up to 74%.

In [22] the application of "*Queuing Models for Large System Migration Scenarios*" is evaluated in "*An Industrial Case Study with IBM System z*". The model is used to predict the performance impact when migrating to a new environment in an industrial context. At first a general modeling methodology is is presented and its application to the migration scenarios is described. The influence of the application performance is predicted with high accuracy.

4 OpenStack Cloud Computing

OpenStack is a fast growing cloud infrastructure as a service solution under heavy development. Its open source approach results in many contributions to the software increasing the flexibility with every release.

4.1 Project Objectives

The OpenStack Project aims "*to produce the ubiquitous Open Source cloud computing platform that will meet the needs of public and private cloud providers regardless of size, by being simple to implement and massively scalable* [23]." The cloud should provide on demand self-services broad network access, rapid elasticity, resource pooling and measured services. It is capable of running private clouds (cloud services only provided for the own organization), public clouds (offering services to others), community clouds (offered to community members and also hosted by the members) and hybrid clouds (cloud is partly public and partly private).

The objective of the OpenStack team is to increase the capabilities of open stack. While at the beginning OpenStack was a mere Infrastructure-as-a-Service application, it has more and more evolved to additionally provide Platform-as-a-Service features which were basically realized in late 2014.

OpenStack is still a young project with the first release (Austin) made public in October 2010 and the first production ready release finalized in September 2011. The development team aims at delivery two releases every year with the current and twelfth release Liberty released in October 2015. With every release new features are added and many projects are currently being developed and competing for their introduction in the stable releases like a DNS service or bare metal provisioning.

4.2 OpenStack Components

A cloud infrastructure consists of multiple modules to achieve the different parts of handling the machines. These modules can be deployed independently on the same node as other modules as well as on dedicated nodes. Some modules can be set up on multiple machines. The module types currently supported by OpenStack are listed below with the name of the OpenStack implementation of this module given in parentheses.

Dashboard (horizon) provides a web-base self-service portal allowing interaction with the underlying services of the cloud environment such as launching an instance, assigning IP addresses and configuring access controls.

Compute (nova) manages the live-cycle of the compute instances provided by the cloud environment and is responsible for the spawning, scheduling and decommission of virtual machines on demand. The compute module is divided into multiple independent sub-modules.

- **API (nova-api)** provides the interface between the other modules and the actual compute nodes.
- **Metadata (nova-metadata)** injects important metadata like public keys to allow secured remote access to the virtual machines.
- **Certification (nova-cert)** provides certificates for the cloud environment to ensure the identity of machines and users.
- **Conductor (nova-conductor)** offers an abstraction layer so no database access by the compute nodes is required.
- **VNC-Proxy (nova-novncproxy)** forwards VNC signals to allow access via remote management using the VNC protocol.
- **Console-Authentication (nova-consoleauth)** ensures the remote environment is only accessible to the users holding the necessary rights and group memberships.
- **Scheduler (nova-scheduler)** schedules the creation and deletion of virtual machines.
- **Actual Compute (nova-compute)** launches and decommissions virtual machines on the node it is running on.

Networking (neutron/nova-network) provides Network-Connectivity-as-a-Service for the other services such as compute. It provides an api allowing the users to define networks and attach machines to them. It can have a plug-able architecture to allow hot plug addition and removal of new machines. Depending on the requirements different Layer 2 and Layer 3 protocols can be supported.

Object Storage (swift) allows to store and retrieve any unstructured data objects. It can offer fault tolerance by duplicating and spreading the storage over multiple nodes. It can usually not be mounted like a file server but instead the exact file to store or retrieve must be specified.

Block Storage (cinder) provides block drives to running instances that can added or removed on demand during operation. It is usually mountable like storage that is directly included in the instance images of the virtual machine.

Identity Service (keystone) provides authentication and authorization services for all the other services. It also provides a catalogue providing users with a list of all available services and their endpoints.

Image Service (glance) stares and retrieves virtual machine disk images that can be used by the compute service to provision new virtual machines.

Telemetry (ceilometer) monitors and meters the cloud environment. The data can be used for billing, benchmarking, scaling and statistical evaluations.

Orchestration (heat) allows the orchestration of composite cloud applications using templates.

Database Service (trove) provides scalable and reliable cloud Database-as-a-Service functionality. Depending on the implementation relational as well as non-relational database engines can be supported.

Data Processing Service (sahara) provides the capability to provision and scale clusters by specifying topology and hardware details.

4.3 OpenStack Architecture

To provide the aforementioned services the modules need to interact to assemble the services from the available sub-services.

Figure 4.1 shows the generalized architecture of the OpenStack cloud infrastructure. On top are the identity, telemetry and dashboard services that interlock with every module in the Cloud Infrastructure. Since they are not necessarily required for the infrastructure to continue running they are depicted outside the system. Around the center the central services are depicted providing object and block storage, image management, network access and compute node provision. Further outside the more abstract services that offer a solution from one source for either orchestration, data processing or database services. It is visible that many services interlock or are chained to provide certain features. It is also visible that certain services like the data processing or database service are just meta services to the base services providing a simpler interface implementing the facade design pattern [24]. This abstraction layer allows users without the knowledge of the base services to create complex infrastructures and services with little effort.

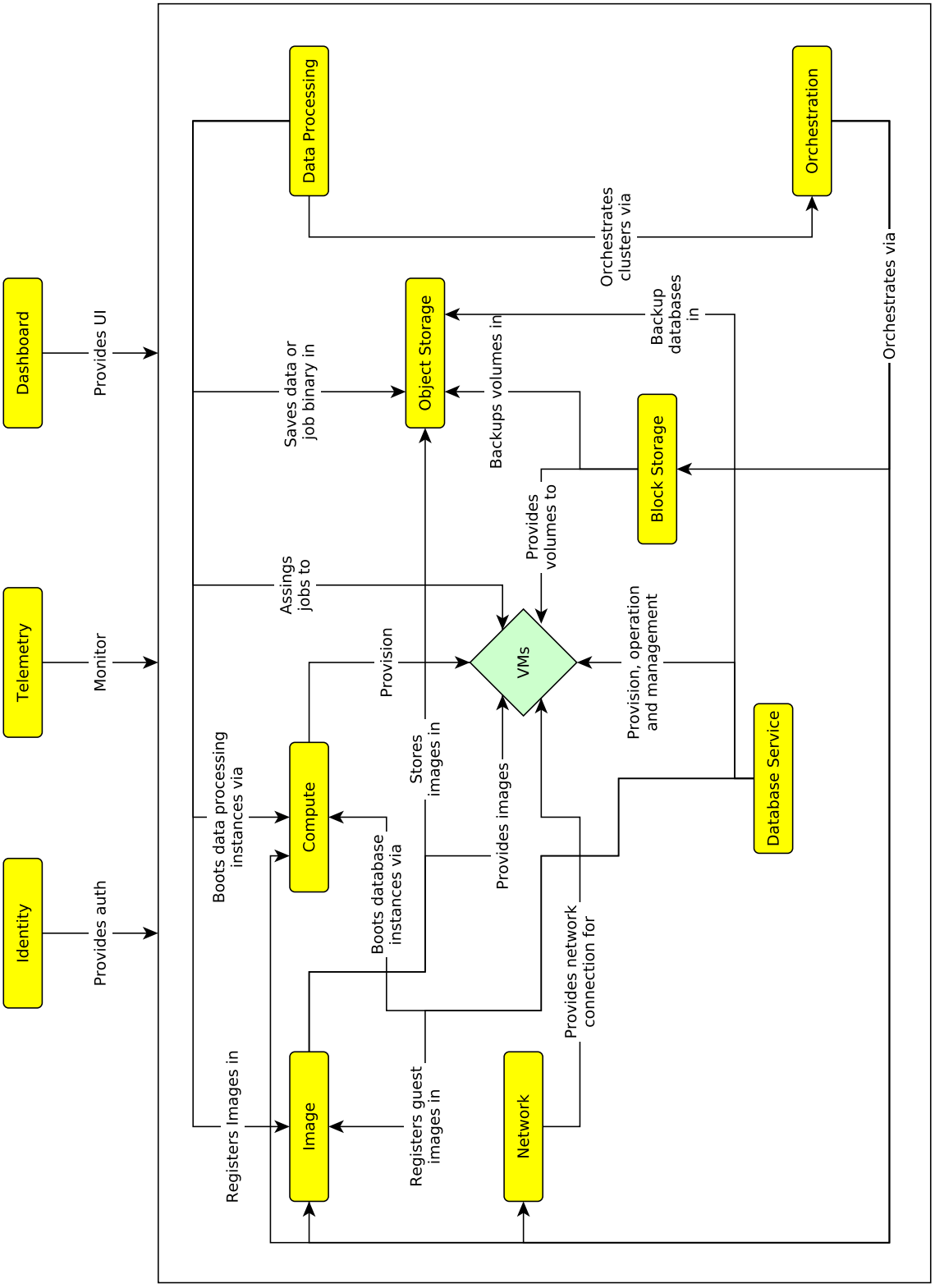

Figure 4.1: Architecture of a cloud infrastructure

5 Testbed Definition for the Measurement of Cloud Migrations

The aim of the testbed is the measurement of the performance of virtual machines and their migrations performance. Therefor in the following an OpenStack testbed is being structured.

5.1 Components in Testbed and Testbed Structure

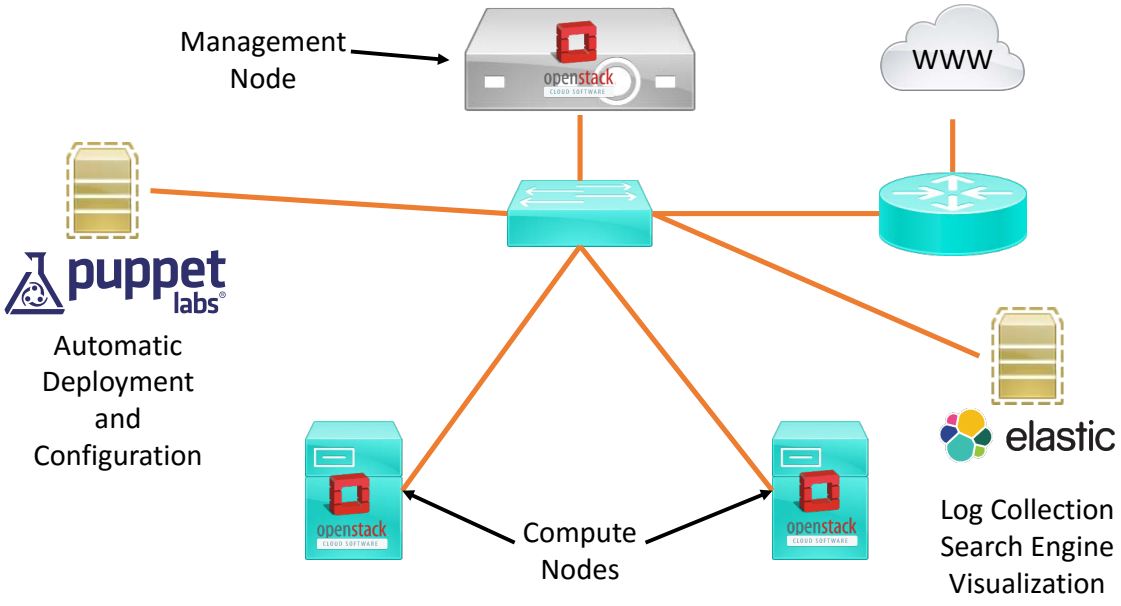

Figure 5.1: Architecture of the used testbed

The testbed consists of multiple nodes as shown in Figure 5.1. A minimal setup of OpenStack, sufficient to create virtual machines with network access as well as perform block and live migrations has been installed. The central management node runs the horizon, nova-api, nova-metadata, nova-cert, nova-conductor, nova-novncproxy, nova-consoleauth, nova-scheduler, glance and keystone services. The compute nodes run the nova-network and the nova-compute service.

Additionally, a node running a Puppet server was configured. It allows the automatic deployment and configuration of applications. Every node except the puppet

server itself is completely configured by puppet allowing the quick replacement of failed machines and fast addition of additional resources (e.g. compute nodes).

A smaller version of the logging infrastructure described in Section 2.5 has been implemented. On every node an instance of Logstash is running as a shipper forwarding log files from the OpenStack services as well as the results of puppet runs and other key system log files to a central Logstash node using a Redis queue as a broker. The Logstash at the central nodes than processes the sent log files indexing the parsed values and grouping events that belong to the same process. This is then forwarded to elastic search as a search cluster solution (in this case the cluster has only one node). For simple access of the indexed data Kibana has been installed to provide a web based graphical user interface with access to the stored information. Kibana supports filtering and the creation of plots for key metrics.

Though the network is only connected to a single NIC per server there a five virtual networks set up. The main network is the public network allowing the communication with the outside. The second network is a network dedicated to the inter process communication of OpenStack. A second network is reserved for a possible future addition of the OpenStack volume service cinder. The third network is used for the puppet deployment. The final network is set aside for the transmission of the logs. This approach allows to define network parameters for each network individually allowing to test the influence of network parameters on OpenStack while still having real time logging capabilities.

For the measurements a script has been developed that performs a selectable number of migrations. The script can be parameterized to which flavor is used, whether live or block migration was selected and which network parameters are desired for a measurement. It can directly set a limit to the total throughput bandwidth, an added latency and an artificial loss rate for network packets.

5.2 Performance Indicators and Metrics

The logging infrastructure collects multiple sets of data from different log files. For each compute node it monitors

- Number of available and used CPUs
- Size of available and used memory
- Size of available and used local storage

From the information sent by the compute nodes the following information is gathered about the virtual machines allowing the key factors to be calculated.

- Date of creating a virtual machine
- Date of destruction
- Date of pausing a virtual machine
- Date of being resumed
- Date of finishing postoperation phase

5.3 Signaling Overhead for Various Migrations

The proposed solution to monitor the migration in the network needs to be evaluated in respect to its efficiency and impact on the network performance. Since the signaling is done over the network the main characteristic is the additional traffic generated by the monitoring solution.

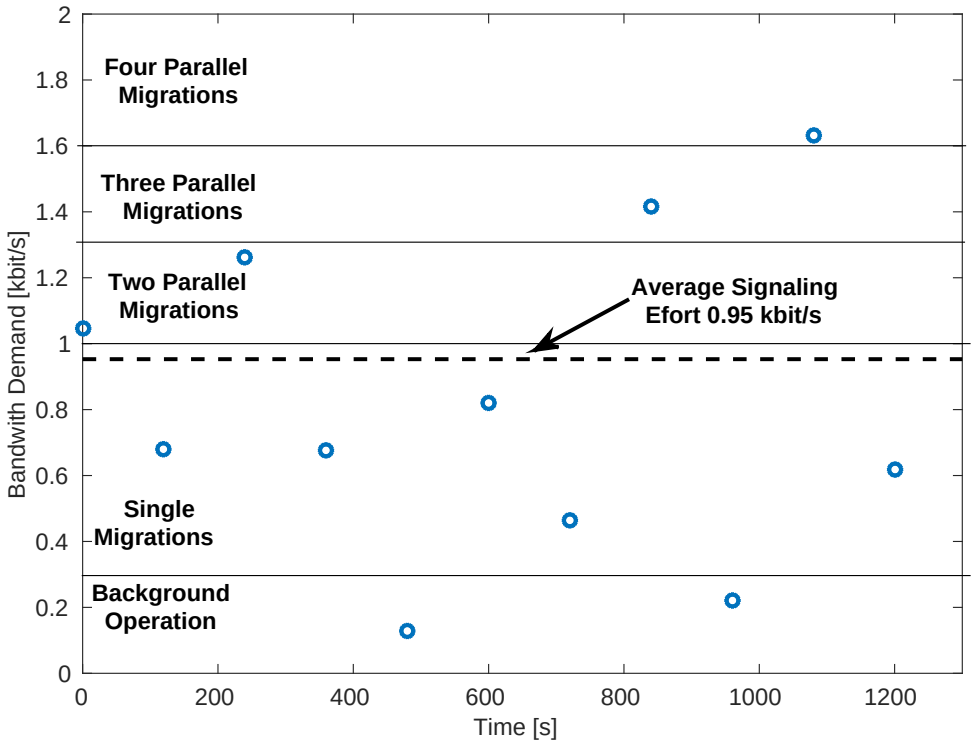

Figure 5.2: Average network load of the monitoring solution in different scenarios

To evaluate the created network load the virtual interface for the the Logstash traffic was captured on the assigned virtual interface using TCPDump[25]. Afterwards offline processing of this dump file has been done using WireShark[26]. The relative time stamp of the transferred packets as well as the size of these packets on the network has been extracted from the logs. Different scenarios where evaluated including migrations of between one and four virtual machines at the same time as well as running only background operations.

Figure 5.2 shows the average bandwidth demand over time. It is visible that the demand is rather low. The background operation only takes about 0.2 kBit/s of bandwidth and even the migration of four machines at once only requires slightly more than 1.6 kBit/s. The average over the complete duration of the different migration scenario is 0.95 kBit/s. Considering connection speeds between data centers usually being above 100 MBit/s and the connections in the data centers are often even faster than that the effect of the monitoring solution on the remaining traffic

should be negligible for a small to medium amount of compute nodes and migrations running. Only in extremely large environment where it is a realistic scenario that thousands of migrations would be running at the same time over dozens of compute nodes a separate link would be required (e.g. switching the Logstash transmissions from a virtual to a dedicated interface).

Therefore the monitoring solution is proven to have little overhead. Real time applications would not be impaired by the parallel transmissions during their normal operation.

6 Performance Assessment of Service Migration Strategies

In the following chapter described testbed is used for the performance assessment of service migration strategies. At first, the different measurement parameters for the evaluation are described. Then, the influence of different network settings and characteristics on the migration performance is quantified in multiple measurements. Subsequently, multiple Internet applications are evaluated for their migratability. At the end of this chapter optimizations are presented to improve that migratability.

6.1 Measurement Parameters for the Evaluation

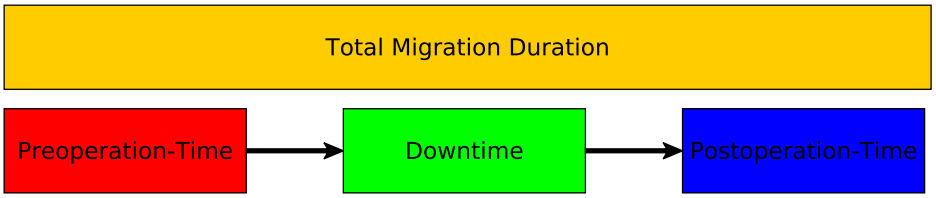

Figure 6.1: Measurement Parameters

Figure 6.1 depicts the different measurement parameters used for the evaluation. Basically the duration of a migration is examined. A migration consists of three major steps. The relevance of each step depends on the requirements that the migrated application and the migrating user pose at the migration process.

Preoperation Time: This time interval starts when the command to initiate the migration is issued. It ends with the point of time at which the time the migrated machine is paused. This includes the handling of the commands via the REST API [27] as well as the synchronization of the file systems on source and target servers, the post-migration generation and transfer of a snapshot of the machine and the allocation of resources on the target system. During this time the machine continues normal operation and is fully accessible through the network.

Downtime: This specifies the time between the pausing of the machine on the source node and the resuming on the target node. It includes the final copying of the latest delta snapshot for block migrations and the terminal synchronization of

the file systems for live migrations. During this time the machine is paused and can neither perform computing operations nor be accessed through the network.

Postoperation Time: This is the time from the point of time when the machine is resumed to the time the operations necessary after resuming are completed. This mainly consists of the reattachment of the network to the machine and the updating of the routes used in the internal network. During this time the machine can perform computing operations but is not guaranteed to be accessible through the network.

As described in the following sections the ratio of these three components varies depending on the external circumstances. It depends on the selected instance flavor as well as on the network parameters and settings.

6.2 Impact of Network Characteristics on the Migration Performance

In the following section the results of the measurements taken for network characteristic influence are presented and discussed. The network settings are modified and measurements for the above specified parameters are taken for multiple types of migration. For each combination of network parameters presented in this chapter at least 15 measurement runs have been performed but in some cases up to 80 repetitions have been done. The amount of runs was determined based on the confidence intervals.

For the evaluation process a virtual image instance of a certain flavor is booted at the testbed based on a CirrOS [28] image. Therefore (if not otherwise stated) the storage of these machines is not artificially filled to the maximum and a lot of free space remains inside the instance. Therefore optimization and compression algorithms have a huge potential.

6.2.1 Impact of Network Throughput Limitations

A huge part of the migration time applies to the transfer of a large amount of data across the network. Consequently we first evaluate the effect of limitations of the throughput rate have on the migrations. Therefore migrations have been measured with bandwidth restriction of 1000 MBit/s (native speed of the used NICs), 100 MBit/s and 10 MBit/s. The measurements have been performed for the m1.tiny, m1.small, m1.medium and m1.large flavors to assess whether the results are applicable across all flavors (The impact of the different flavors is detailed and discussed in Section 6.2.4).

At first the the table showing the relevant factors is presented. Then the total duration is analyzed followed by the preoperation time, the downtime and postoperation time. At the end of the subsection a short summary is drawn.

Migration Type	Flavor m1.	Speed [MBit/s]	avg. Duration [s]	avg. Preoptime [s]	avg. Downtime [s]	avg. Postoptime [s]
block	tiny	10	76.606 ± 5.324	74.580 ± 5.305	0.403 ± 0.075	1.623 ± 0.122
block	tiny	100	14.200 ± 0.380	12.372 ± 0.354	0.188 ± 0.060	1.638 ± 0.123
block	tiny	1000	12.333 ± 0.712	10.521 ± 0.694	0.214 ± 0.074	1.598 ± 0.127
block	small	10	119.284 ± 3.917	117.307 ± 3.956	0.388 ± 0.055	1.588 ± 0.085
block	small	100	18.513 ± 0.401	16.725 ± 0.374	0.181 ± 0.062	1.606 ± 0.115
block	small	1000	14.443 ± 0.965	12.628 ± 0.865	0.228 ± 0.089	1.587 ± 0.154
block	medium	10	172.856 ± 0.743	170.847 ± 0.778	0.393 ± 0.110	1.616 ± 0.121
block	medium	100	24.375 ± 0.352	22.571 ± 0.326	0.206 ± 0.066	1.598 ± 0.178
block	medium	1000	18.045 ± 1.004	16.138 ± 0.900	0.224 ± 0.074	1.683 ± 0.155
block	large	10	271.278 ± 0.417	269.373 ± 0.375	0.352 ± 0.041	1.554 ± 0.137
block	large	100	34.446 ± 0.806	32.656 ± 0.676	0.248 ± 0.078	1.541 ± 0.115
block	large	1000	23.579 ± 0.983	21.766 ± 0.863	0.189 ± 0.061	1.624 ± 0.166
live	tiny	10	65.628 ± 1.963	63.478 ± 5.305	0.390 ± 0.094	1.760 ± 0.105
live	tiny	100	12.747 ± 0.295	10.940 ± 0.354	0.134 ± 0.014	1.673 ± 0.136
live	tiny	1000	10.726 ± 0.423	8.967 ± 0.694	0.127 ± 0.023	1.630 ± 0.070
live	small	10	107.791 ± 1.598	105.725 ± 3.956	0.414 ± 0.140	1.652 ± 0.128
live	small	100	16.915 ± 0.401	15.149 ± 0.374	0.188 ± 0.100	1.578 ± 0.094
live	small	1000	13.658 ± 0.499	11.921 ± 0.865	0.107 ± 0.022	1.630 ± 0.057
live	medium	10	163.559 ± 0.647	161.413 ± 0.778	0.369 ± 0.061	1.777 ± 0.102
live	medium	100	22.575 ± 0.332	20.797 ± 0.326	0.222 ± 0.082	1.556 ± 0.102
live	medium	1000	16.393 ± 0.454	14.609 ± 0.900	0.140 ± 0.036	1.643 ± 0.069
live	large	10	264.068 ± 3.818	261.961 ± 0.375	0.406 ± 0.080	1.702 ± 0.148
live	large	100	32.670 ± 0.348	30.789 ± 0.676	0.228 ± 0.042	1.653 ± 0.121
live	large	1000	22.389 ± 0.381	20.566 ± 0.863	0.154 ± 0.015	1.668 ± 0.066

Table 6.1: Migration duration sorted by flavor with ascending throughput limits with confidence intervals with 95% confidence level

Table 6.1 shows the results of the measurements taken including confidence intervals with 95% confidence level. The migration time increases when the throughput is reduced. Therefore the initial assumption that the limitation of the throughput influences the migration time is proven correct.

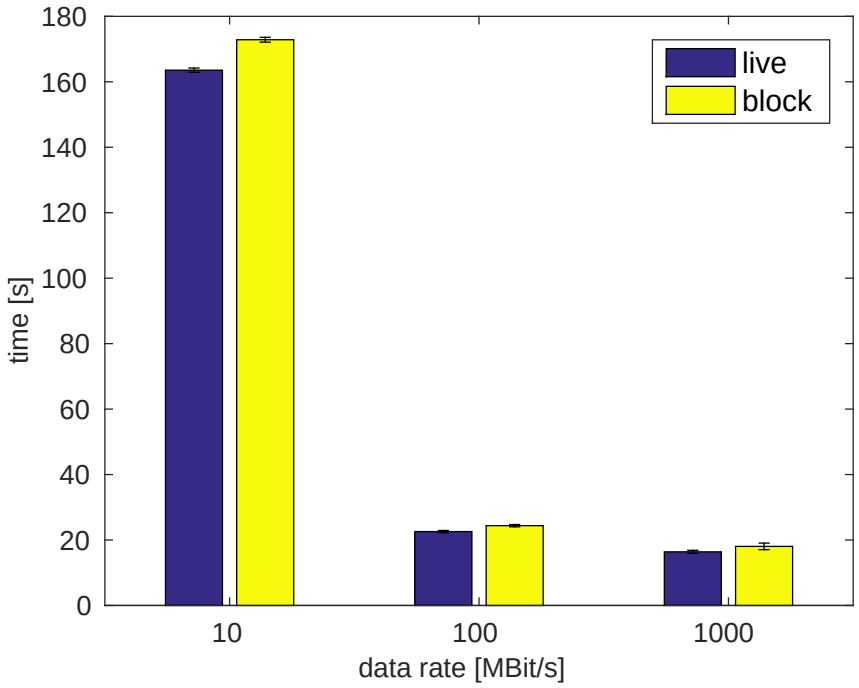

Figure 6.2: Migrations of the medium flavor using different throughput limits

Figure 6.2 shows the migration times for the medium flavor. As can be seen in Table 6.1 these results are representative for the other flavors. Between 100 MBit/s and 1000 MBit/s the difference is quite small compared to the difference between 100 MBit/s and 10 MBit/s, though both differ at a factor of ten. This leads to the assumption that - especially for the smaller flavors - a large factor of the time spent on the migration is not at all or only slightly dependant on network throughput. This includes Memory allocation, the booting, pausing and resume processes of the virtual machines, latency critical low throughput applications like the messaging queues or the REST API used by OpenStack[27]. The factor affected the most is the copying of the machine instance.

The average migration times of the live migration approach lie always below to the block migration approach. While for the highest throughput level of 1000 MBit/s the confidence intervals still overlap for some flavors, the increase is significant as the throughput decreases.

As can be seen in Table 6.1 the majority of the time spent on the migration is always spent on the preoperation time. Therefore it is considerably affected by the throughput limit. As shown in Figure 6.3 the preoperation time is largely increased when reducing the available bandwidth. Like the total duration of the migration the

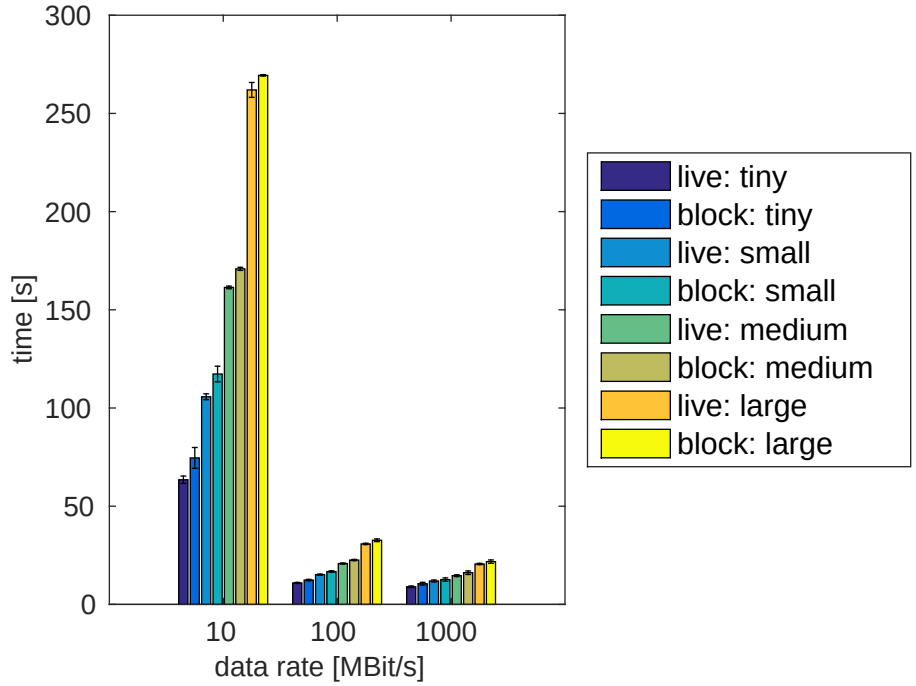

Figure 6.3: Preoperation times for various flavors and migration types grouped by throughput limit

preoperation time of the live migration is shorter than for the block migration with the advantage becoming more significant with lower throughput. This is explainable due to the fact, that one major factor of the preoperation phase is moving a snapshot of the virtual machine from the source to the target server in block mode respectively synchronizing the source machine with the central storage and that again with the target machine. This process mainly consists of transferring a large amount of data over the network and is therefore very sensitive to bandwidth limitations.

Figure 6.4 shows the downtimes from Table 6.1. The downtimes are slightly sensible to the change of the throughput limits but the effect is very weak compared to the preoperation times. This is due to the fact that because only a small part of the downtime phase relies on network transport and consequently the bandwidth limitations have little effect. Only the final sync of the storage (live migration) respectively the transmission of the final delta snapshot (block migration) is performed over the network. Other tasks like pausing the machine on the source node and resuming it on the target node are not dependant on the network throughput.

It is visible that at higher speeds the live migration approach keeps the downtimes on average up to half as short as the block migration approach. The latter has a large confidence interval, so while slower on average there are single occurrences where the block migration has a shorter downtime. For computational intensive applications that might be a critical advantage for the live-migration since the interruption of the computing process is only interrupted for a short time.

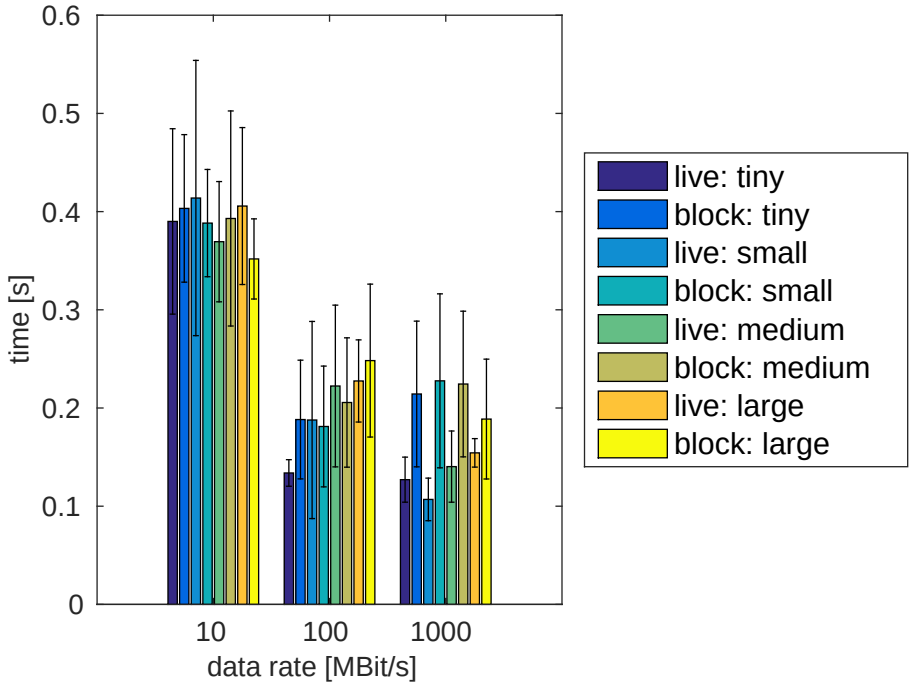

Figure 6.4: Downtimes for various flavors and migration types grouped by throughput limitation

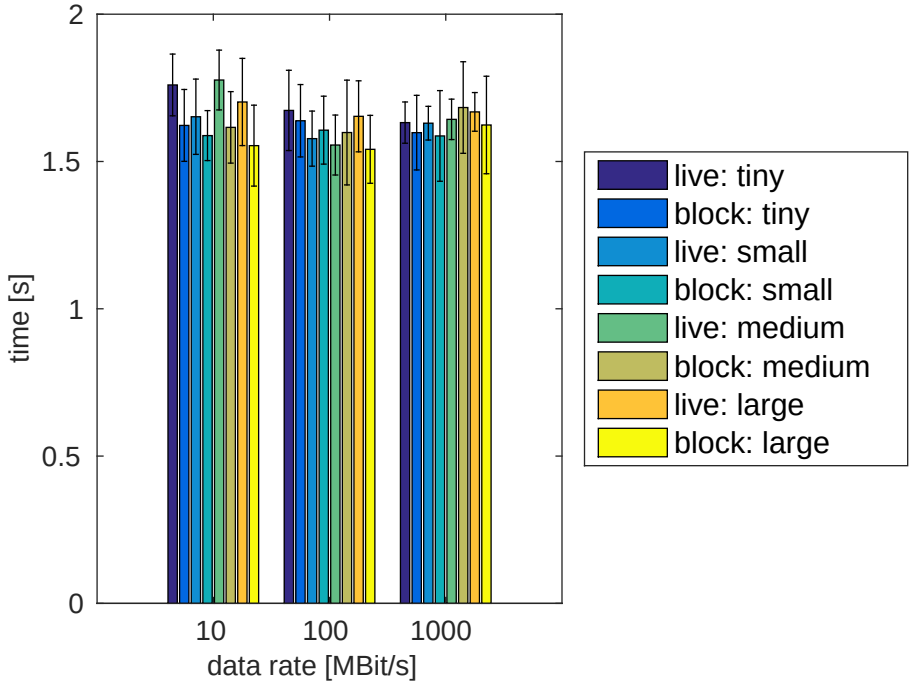

Figure 6.5: Postoperation times for various flavors and migration types grouped by throughput limit

In Figure 6.5, the postoperation times from Table 6.1 are visualized. From the depicted values and confidence intervals no effect of the throughput limits on the postoperation times can be proven beyond reasonable doubt. There is no significant difference between live migration and block migration. Therefore the effect of throughput limitations on the postoperation time is negligible.

To sum up, the duration of a migration is highly dependent on the network speed. The major part of the duration and also its increase is provided by the preoperation time. For this factor and the total duration the live migration is always superior to the block migration. The downtime is slightly sensitive to the available network bandwidth. For high throughput values the live migration is faster than the block migration but that effect is no longer significant for slower network speeds. The postoperation time is not affected at all by the limitations and the different migration types.

6.2.2 Impact of Network Delays on the Migration Performance

Migrations require communications between the involved nodes. Those vary from REST API requests over message queue and interaction packets to the movement of the virtual machine snapshots. The amount of communication requests varies between the different migration strategies and therefore we evaluate the effect of increased network latencies on those communications. Thus, migrations have been measured for no extra delay, 10 ms, 50 ms and 100 ms delay. The measurements have been performed for the m1.tiny, m1.small, m1.medium and m1.large flavors to assess whether the results are applicable across all flavors.

Table 6.2 shows the results of the measurements taken including confidence intervals with 95% confidence level. For increasing network delay the migration time increases for all flavors. Therefore the assumption of the negative effect of increased network delay on the migration duration is proven correct.

Figure 6.6 shows the migration times under different latency settings for the tiny flavor. As can be seen in Table 6.2 these results are representative for the other flavors. The difference between 0 ms and 10 ms delay is rather small while the duration increase for latencies up to 50 and 100 ms is almost linear to the delay increase. As discussed in Subsection 6.2.1, this can be explained by the fact that that a lot of operations are performed that are not dependant on the network performance. Also the mere transfer of the snapshot requires only a few connections so it is only slightly affected. The increase between 10 and 100 mill seconds leads to the assumption that the latency can probably be modelled by a first or second degree polynomial function.

Contrary to the different throughput limits evaluated in Subsection 6.2.1 the live migration loses its advantage in many cases. Was it faster for every bandwidth limit in the last section, now it is only faster at no extra latency. At ten ms the block migration brakes even with a slight advantage that significantly increases with growing latency. The same applies for the other flavors but the relative advantage for the block migration at high latencies is slightly smaller due to the higher amount

Migration Type	Flavor m1.	Delay [MBit/s]	avg. Duration [s]	avg. Preoptime [s]	avg. Downtime [s]	avg. Postoptime [s]
block	tiny	0	7.531 ± 00.116	5.913 ± 00.107	0.253 ± 0.025	1.365 ± 0.054
block	tiny	10	10.894 ± 00.321	8.581 ± 00.437	0.308 ± 0.067	2.005 ± 0.204
block	tiny	50	33.074 ± 02.396	27.778 ± 02.921	1.101 ± 0.313	4.195 ± 1.125
block	tiny	100	56.068 ± 02.590	45.924 ± 02.331	1.222 ± 0.768	8.922 ± 1.656
block	small	0	8.047 ± 00.193	6.457 ± 00.191	0.280 ± 0.063	1.310 ± 0.084
block	small	10	11.569 ± 00.558	9.654 ± 00.474	0.226 ± 0.078	1.688 ± 0.274
block	small	50	35.383 ± 02.954	30.326 ± 03.238	0.505 ± 0.261	4.552 ± 0.754
block	small	100	62.779 ± 03.741	53.576 ± 03.688	1.112 ± 0.664	8.091 ± 0.936
block	medium	0	8.741 ± 00.182	7.075 ± 00.185	0.299 ± 0.034	1.367 ± 0.029
block	medium	10	12.910 ± 00.823	10.598 ± 00.886	0.332 ± 0.086	1.980 ± 0.202
block	medium	50	41.581 ± 03.707	36.347 ± 03.830	0.652 ± 0.272	4.583 ± 0.796
block	medium	100	73.490 ± 04.260	64.120 ± 04.004	1.426 ± 0.801	7.944 ± 1.322
block	large	0	10.241 ± 00.274	8.594 ± 00.235	0.302 ± 0.049	1.345 ± 0.079
block	large	10	14.342 ± 00.526	12.133 ± 00.577	0.321 ± 0.073	1.888 ± 0.241
block	large	50	52.206 ± 04.972	46.664 ± 05.114	0.793 ± 0.247	4.750 ± 0.524
block	large	100	86.144 ± 10.469	76.539 ± 10.288	1.420 ± 0.737	8.185 ± 1.392
live	tiny	0	6.829 ± 00.237	5.076 ± 00.228	0.271 ± 0.049	1.482 ± 0.055
live	tiny	10	11.864 ± 00.423	9.322 ± 00.325	0.263 ± 0.159	2.279 ± 0.454
live	tiny	50	38.402 ± 02.483	32.889 ± 02.496	0.372 ± 0.093	5.141 ± 1.163
live	tiny	100	69.905 ± 07.106	60.089 ± 07.093	0.809 ± 0.515	9.007 ± 1.120
live	small	0	7.524 ± 00.204	5.779 ± 00.213	0.260 ± 0.059	1.485 ± 0.058
live	small	10	12.790 ± 00.613	10.216 ± 00.589	0.187 ± 0.092	2.386 ± 0.140
live	small	50	44.869 ± 03.322	39.080 ± 03.111	0.430 ± 0.058	5.359 ± 0.961
live	small	100	79.181 ± 04.538	69.265 ± 05.341	0.665 ± 0.071	9.251 ± 1.798
live	medium	0	8.228 ± 00.209	6.415 ± 00.219	0.343 ± 0.062	1.469 ± 0.029
live	medium	10	13.618 ± 00.628	11.043 ± 00.596	0.203 ± 0.030	2.371 ± 0.278
live	medium	50	49.298 ± 04.133	42.721 ± 04.234	0.519 ± 0.241	6.059 ± 1.577
live	medium	100	84.653 ± 06.314	75.539 ± 06.050	0.677 ± 0.087	8.437 ± 1.615
live	large	0	9.417 ± 00.207	7.686 ± 00.198	0.341 ± 0.051	1.390 ± 0.086
live	large	10	16.069 ± 01.568	13.710 ± 01.662	0.191 ± 0.035	2.168 ± 0.365
live	large	50	56.299 ± 08.190	50.608 ± 08.585	0.417 ± 0.117	5.274 ± 1.321
live	large	100	96.000 ± 12.398	86.239 ± 11.989	0.585 ± 0.201	9.176 ± 1.411

Table 6.2: Migration durations sorted by flavor with ascending network packet delay with confidence intervals with 95% confidence level

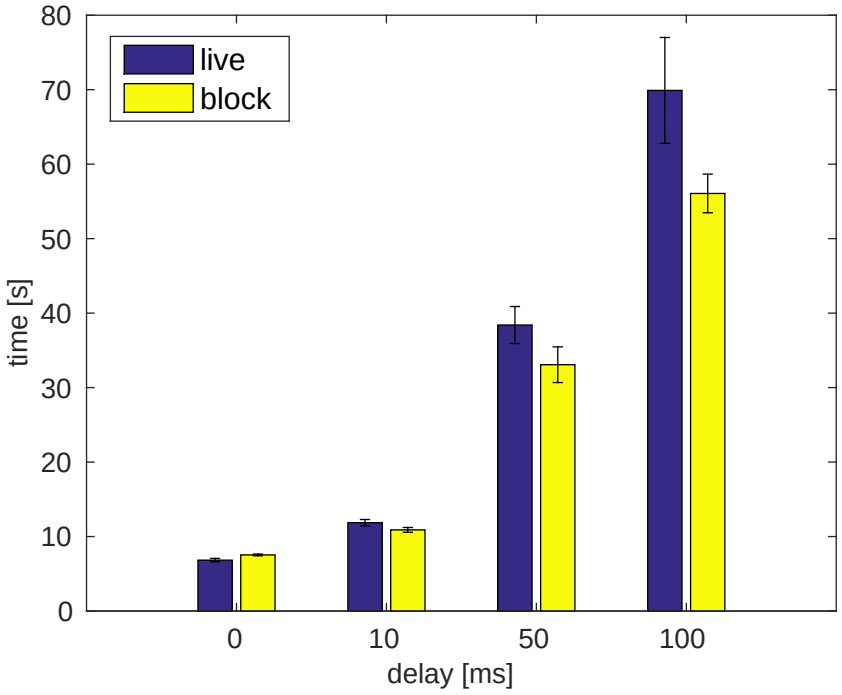

Figure 6.6: Migrations of virtual instances with the tiny flavor using different latencies

of snapshot data moved that is less vulnerable to network delay.

From Table 6.2, it is visible that the major component of the migration duration is again the preoperation time. It is significantly increased by the additional delay as shown in Figure 6.7. Similar to the total duration, the preoptime is smaller for the live migration without added latencies but the block migration already catches up and becomes faster at about 10 ms. This is probably related to the fact, that the live migration requires permanent connections between the central storage node and the compute nodes. Over this connections many smaller requests are made when storing the changes of the instance to the central node and when retrieving the files to restore the machine on the target node. The block migration on the other hand usually transfers bigger chunks of data, namely the snapshots of the machines, having fewer requests than the live migration and therefore less affected by the increased delay. The benefit of the block migration increases with higher added latency and for most flavors exceeds the range where the confidence intervals overlap. Like the total duration, the increase above 10 ms is nearly linear which hints at the possibility to model this effect with a simple polynomial function.

Though a relatively small part of the migration the downtime is important for computationally intensive application being the only part the actual computing process stops. Figure 6.8 presents the values listed in Table 6.2 depicting the effect of increased latency on the downtime. The effect is relatively weak in contrast to the preoperation time but still significant. From 0 to 10 ms of additional latency there

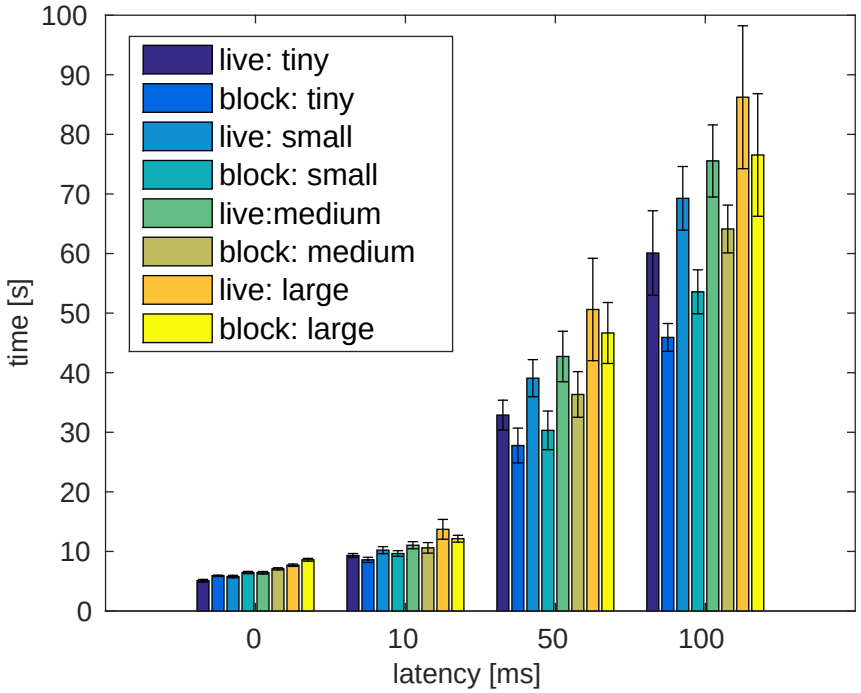

Figure 6.7: Preoperation times for various flavors and migration types grouped by latency

is little to none increase in downtime. In some cases the downtime even decreases. This is probably due to optimizations in the OpenStack migration algorithm adapting for high or low latency networks. At higher latencies the downtime is creased up to a factor of over two.

Contrary to the total duration and the preoperation time, the downtime favors the live migration at higher latencies. The live migration still violates the 0.5 second limit desired by the OpenStack standard configuration but is on average faster than to the block migration. This can be explained by the fact that during the downtime operation of the live migration the network shares are already synchronized while for the block migration the transfer of the final delta snapshot is still required. Additionally, the downtimes for the block migration are extremely varying explaining the large confidence intervals. This is due to the fact that depending on the machine operation (e.g. CirrOS performs logging events, etc.) the amount of data transferred in the final delta snapshots varies and sometimes there is no need to transfer such a snapshot at all.

The large jump for the tiny flavor at the step between 10 ms and 50 ms is possibly accountable to a different optimization approach for the duration taking effect. This would correlate with the relatively strong advantage of the block migration for the duration with the tiny flavor. To model the downtime phase of this migration with a high level of confidence, a detailed understanding of the internal mechanisms of the different migration types is required.

While throughput limitations had no measurable effect on the postoperation time

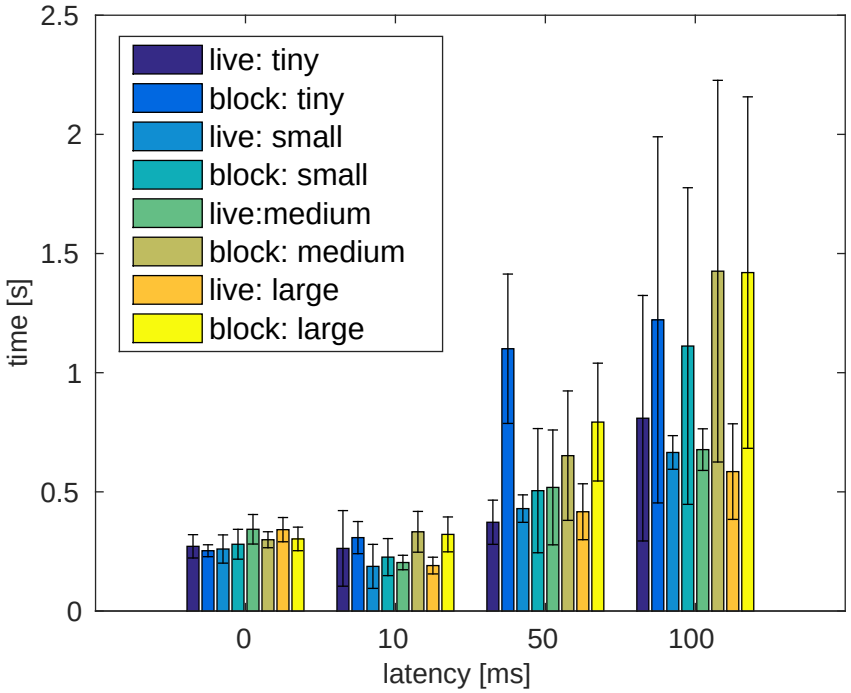

Figure 6.8: Downtimes for various flavors and migration types grouped by latency

nothing similar can be said in terms of additional network delay. The postoperation time scales nearly linear with the increased delay as shown in Figure 6.9 again using the values from Table 6.2. This effect occurs because the major task during the postoperation time is the reestablishment of the network connections to the virtual machine at the target host. This usually requires a large number of small consecutive requests over the network that do not require a lot of bandwidth but are impaired by the added delays. Since the postoperation phase is very similar for block and live migration like for the throughput limits, no significant difference between block and live migration can be proven beyond reasonable doubt.

Summing up, the migration duration strongly rises when increasing the network delays at an almost linear manner with nearly the same factor as the delay increase. Like for the throughput limitations the major part of the migration duration as well as its increase is provided by the preoperation time. The fashion of the increase as well as the factor are similar to the total duration. Regarding the latency better overall durations of the live migration are no longer present. While at no extra latency the live migration is still faster, the block migration catches up and becomes faster than the live migration with rising network delay. The downtime is affected by the increased latency to smaller extend than the preoperation time. But here the live migration is still quite faster than the block migration. The postoperation time is increased in a matter similar to the total duration. Here, no discernible advantage for one of the migration types is visible.

Unlike the bandwidth limitation, for the latencies, it is harder to give a general recommendation for the preferable migration mode in networks with higher latency. We recommend to use block migration for the movement of machines where the

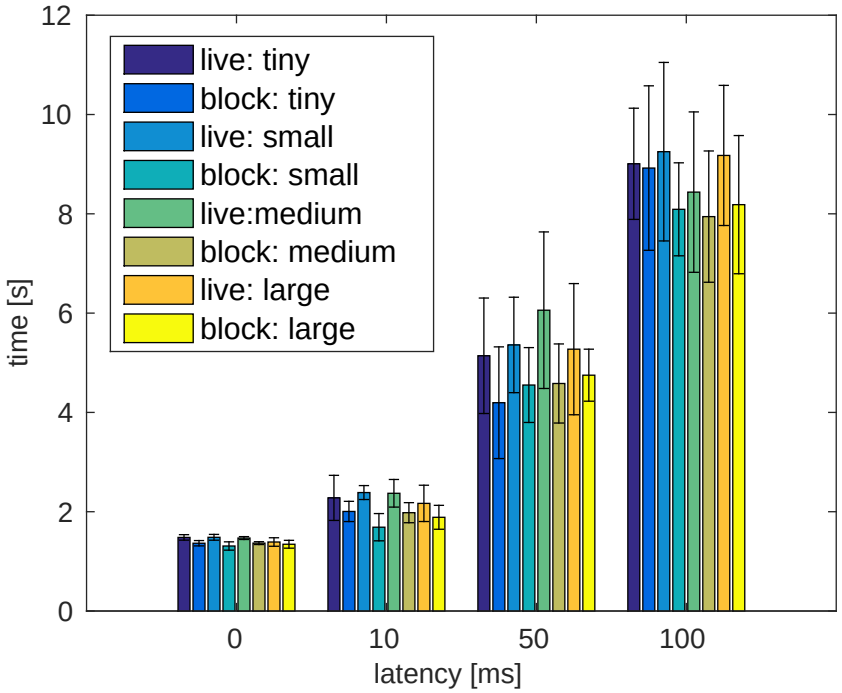

Figure 6.9: Postoperation times for various flavors and migration types grouped by latency

total duration of the migration is important (e.g. to get them off dying machines or machines that suffer a very bad connection to the outside network). For machines where the continuous computation is more important (e.g. machines that continuously compute results from data which are periodically received over the network) the live migration is more recommendable due to the shorter downtime. For machines that require a short postoperation time the only solution would be switching the network between source, controller and target host for a network with shorter delays.

6.2.3 Packet Loss in the Network and Related Effects

A third network parameter is the percentage of packets that are not successfully transferred due to uncorrectable bit errors in the line. This percentage is called the packet drop rate. The effect of the drop rate depends on the used algorithms and the network protocol. Single and multiple acknowledgments can make a difference as well as whether successful packets after the first lost packet have to be re-sent or not like in TCP's selective acknowledgment [29]. In an ideal scenario where only the files that are dropped have to be re-sent and no ACK transmissions are lost the required amount of data would be calculated by the formula in Equation 6.1 where n is the factor of required data transferred and d is the drop rate.

$$n = \sum_{i=0}^{\infty} d^i \tag{6.1}$$

OpenStack uses different approaches in different modules making use of UDP or TCP. Therefore measuring the effect of different loss rates is required to estimate the effect of unstable links on the different subtasks of the migration duration. Thus migrations have been measured without error as well as for one and five percent drop rate. Higher drop rates have not been measured due to a rate of five percent already being very high and cases where higher rates are the case would certainly not occur in a data center. Links between two data center with such a loss rate would probably be rerouted on a more stable path. The measurements have been performed for the m1.tiny, m1.small, m1.medium and m1.large flavors to assess whether the results are applicable across all flavors (The available flavors are detailed in Subsection 6.2.4). For one percent packet loss the ideal factor would be 1.0101 according to Equation 6.1 and for five percent packet loss it would increase to 1.05263.

Table 6.3 shows the results of these measurements including confidence intervals with 95% confidence level. For increasing drop rate in the network the migration duration increases significantly throughout all flavors and migration types. The assumption that the drop rate impairs the migration process is therefore proven correct.

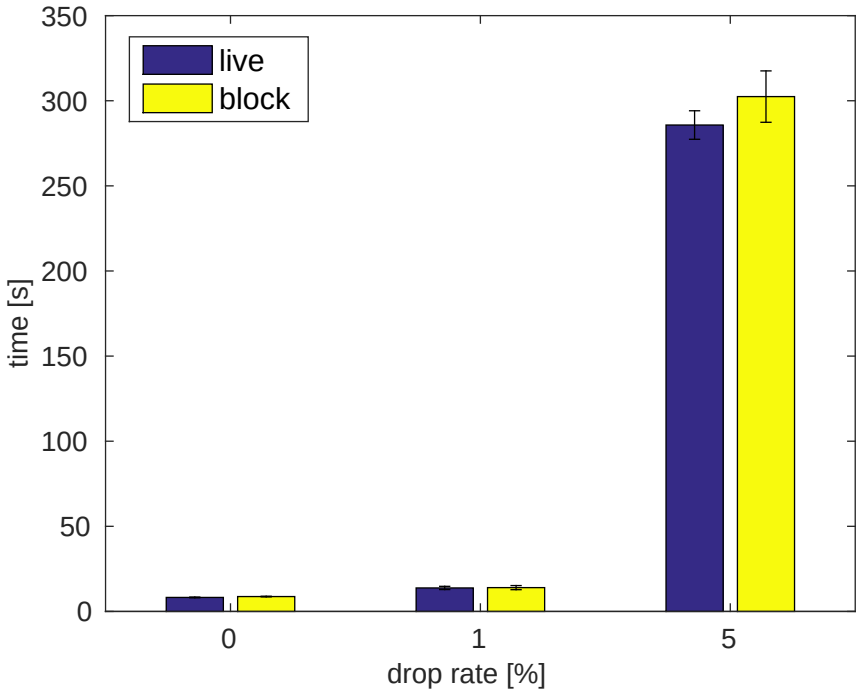

Figure 6.10: Migrations of the medium flavor using different drop rates

Figure 6.10 exemplarily shows the migration times for the medium flavor with different drop rates. The values were taken from Table 6.3 where it can also be seen that the medium flavor is representative for the other tested flavors which have similar patterns with only the factors diverging. The increase between no and one percent drop rate varies between 38 % for the tiny flavor and 77 % for the large

Migration Type	Flavor m1.	Drop Rate	avg. Duration [s]	avg. Preoptime [s]	avg. Downtime [s]	avg. Postoptime [s]
block	tiny	0	7.531 ± 00.116	5.913 ± 00.107	0.253 ± 0.025	1.365 ± 0.054
block	tiny	1	10.403 ± 00.900	8.495 ± 01.021	0.400 ± 0.171	1.508 ± 0.201
block	tiny	5	131.591 ± 08.414	128.283 ± 08.510	0.822 ± 0.457	2.485 ± 0.706
block	small	0	8.047 ± 00.193	6.457 ± 00.191	0.280 ± 0.063	1.310 ± 0.084
block	small	1	12.118 ± 00.555	10.191 ± 00.545	0.379 ± 0.173	1.548 ± 0.178
block	small	5	198.495 ± 20.971	194.992 ± 20.814	1.493 ± 0.882	2.010 ± 0.542
block	medium	0	8.741 ± 00.182	7.075 ± 00.185	0.299 ± 0.034	1.367 ± 0.029
block	medium	1	13.988 ± 01.209	11.851 ± 01.321	0.709 ± 0.284	1.427 ± 0.112
block	medium	5	302.444 ± 15.093	298.781 ± 15.550	1.535 ± 1.324	2.128 ± 0.731
block	large	0	10.241 ± 00.274	8.594 ± 00.235	0.302 ± 0.049	1.345 ± 0.079
block	large	1	18.116 ± 01.070	16.294 ± 01.229	0.379 ± 0.150	1.442 ± 0.184
block	large	5	475.463 ± 10.323	471.767 ± 10.470	1.243 ± 0.961	2.453 ± 0.634
live	tiny	0	6.829 ± 00.237	5.076 ± 00.228	0.271 ± 0.049	1.482 ± 0.055
live	tiny	1	9.922 ± 00.745	7.808 ± 00.619	0.443 ± 0.273	1.671 ± 0.281
live	tiny	5	111.702 ± 06.337	107.973 ± 06.263	1.106 ± 0.541	2.244 ± 0.367
live	small	0	7.524 ± 00.204	5.779 ± 00.213	0.260 ± 0.059	1.485 ± 0.058
live	small	1	11.097 ± 00.797	9.195 ± 00.731	0.156 ± 0.063	1.746 ± 0.154
live	small	5	185.295 ± 09.448	182.002 ± 09.820	1.071 ± 0.826	2.222 ± 0.466
live	medium	0	8.228 ± 00.209	6.415 ± 00.219	0.343 ± 0.062	1.469 ± 0.029
live	medium	1	13.804 ± 00.948	11.769 ± 00.894	0.395 ± 0.255	1.640 ± 0.144
live	medium	5	285.740 ± 08.375	282.120 ± 08.069	1.047 ± 0.611	2.574 ± 0.636
live	large	0	9.417 ± 00.207	7.686 ± 00.198	0.341 ± 0.051	1.390 ± 0.086
live	large	1	17.012 ± 01.053	15.005 ± 01.058	0.463 ± 0.171	1.544 ± 0.141
live	large	5	452.638 ± 13.102	449.487 ± 12.706	0.914 ± 0.519	2.238 ± 0.589

Table 6.3: Migration duration sorted by flavor with ascending packet drop rate with confidence intervals with 95% confidence level

flavor in the block migration. The increase between one and five percent drop rate is even larger. The duration rises between 1164 % for the tiny flavor and 2525% for the large flavor. Any of these factors is significantly larger than the optimal factors calculated above which leads to the conclusion that a lot of packets are redundantly transferred and the link does not operate at optimal speed. The increase is more than linear and modelling would at least require a quadratic function.

Independently of the drop rate the live migration is slightly superior to the block migration in the characteristic of duration. While the relative advantage is constant the absolute advantage increases with instance flavor and drop rate culminating at almost 23 seconds for the large flavor at five percent drop rate.

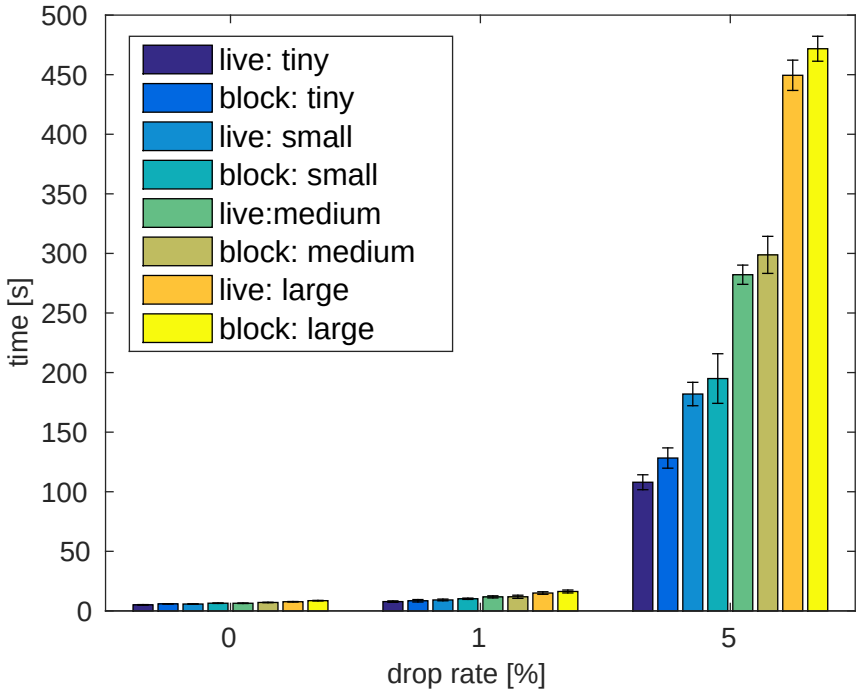

Figure 6.11: Preoperation times for various flavors and migration types grouped by drop rate

As in the previous measurements for throughput limitations and delay once again the preoperation time is the largest part of the total migration duration. Figure 6.11 depicts the values and confidence intervals from Table 6.3. Similar to the duration the live migration has a slight advantage to the block migration. The factor the preoperation time increases with the drop rate is even higher than for the total duration. This leads to the assumption that the part influenced most by the packet loss is the transmission of the snapshot for the block migration and the synchronization of the network file system for the live migration. Like for the duration the increase is more than linear but can probably be resembled by a simile polynomial function.

Figure 6.12 shows the development of the downtime over the different network loss

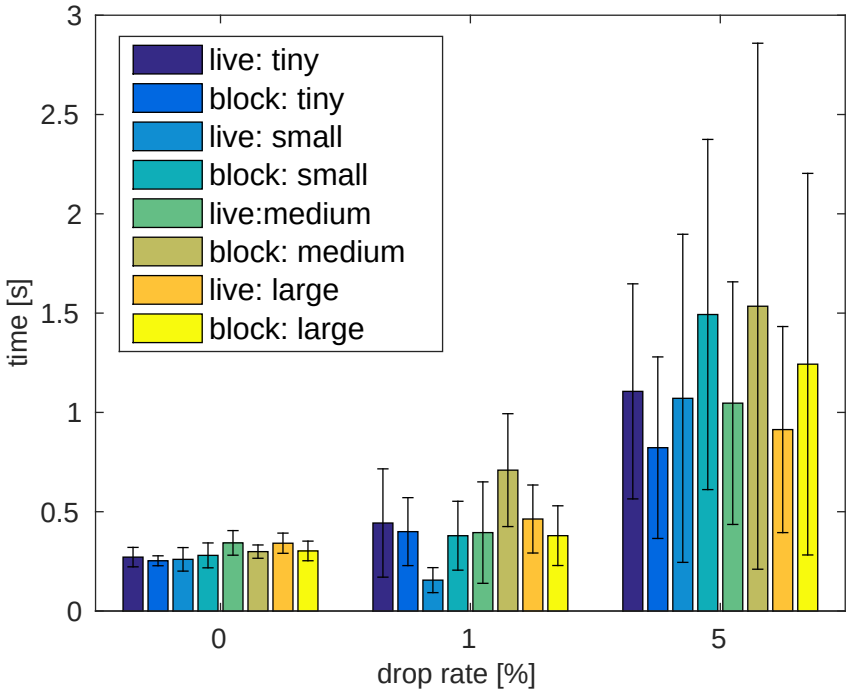

Figure 6.12: Downtimes for various flavors and migration types grouped by drop rate

quotas using the values from Table 6.3. Ignoring singular exceptions the average down time increases slowly with the drop rate. For the first step the only few scenarios (e.g. block medium) reach and exceed the factor of two. A slightly larger increase visible at the second step with all flavors and modes at least doubling.

Unlike for the throughput and latency measurements there is no clear winner between block and live migration. On average the live migration is slightly superior except for the tiny flavor but the confidence intervals are so large that the advantage can not be confirmed beyond reasonable doubt. A modelling of the influence would again require detailed modelling of the internal algorithms since the recorded values don't suggest any simple function to follow.

The behavior of the postoperation time is less sensitive to the increased drop rate. Figure 6.13 shows the postoperation times from Table 6.3. There is a slight increase between no and one percent drop rate and a slightly larger increase when increasing the rate to five percent. The increase is very small at about sixty percent between zero and five percent drop rate. Thus a modeling could be possibly done by a polynomial function with small factors. As in the previous subsections for the drop rate there is no significant advantage of a migration mode that can be proven beyond reasonable doubt.

To sum up, the migrations are very sensitive to increases in the packet loss rate. The increases affect the total duration as well as the preoperation time in a more than linear manner to a high degree. Like for the throughput limitations but unlike the network delays, the live migration is faster over all drop rate measurements. The downtime is slightly influenced by the increase in packet loss. Unlike the other

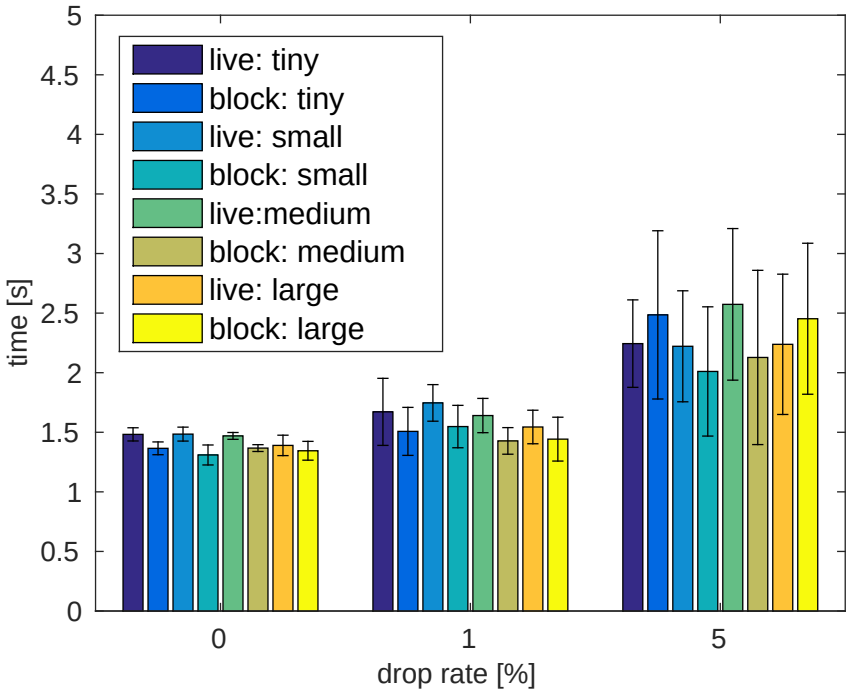

Figure 6.13: Postoperation times for various flavors and migration types grouped by drop rate

two measurement series, there is no clear advantage for the live migration when it comes to the downtime aspect. The postoperation time is the least sensitive to the increase in drop rate. Like with the other two experiments there is no advantage of any type of migration.

When considering the choice of migration type in an environment with high packet drop rate, live migration has a slight advantage and would be preferable. Since the effect of data loss is very strong on the total duration the allocation of resources to increase the link quality for this metric would be recommendable.

6.2.4 Impact of Instance Flavor on the Migration Performance

OpenStack provides five basic types of flavors as seen in Table 6.4. The flavors are usually distinguished by the amount of main memory they allocate, the amount of disk space they require and the number of virtual central processing units (VCPUs) they utilize. Within the testbed measurements for the flavors m1.tiny, m1.small, m1.medium and m1.large have been performed. No measurements for the m1.xlarge flavor were possible since the host machines only run on 16 GB main memory and with the memory needed by the host operating system not enough memory for an xlarge instance remains.

In the following the influence of the instance flavor will be shown at first the effect will be measured with fixed throughput limits and afterwards with a fixed network delay.

Name	Memory[MiB]	Disk [GiB]	VCPUs
m1.tiny	512	1	1
m1.small	2048	20	1
m1.medium	4096	40	2
m1.large	8192	80	4
m1.xlarge	16384	160	8

Table 6.4: Default flavors provided by OpenStack

Fixed throughput limit

Table 6.1 used in Subsection 6.2.1 also shows the measured duration, preoperation, postoperation and downtimes for different flavors for every throughput setting. In Section A the values from Table 6.1 can be found again sorted by speed instead of flavor in Table A.1.

The effect of the flavor on the duration is exemplary shown for the fixed throughput of 100 MBit/s in Figure 6.14. It is visible that the instance size has in fact some influence on the migration duration. The effect is less than either disk size, memory size or CPU number difference. This leads to the assumption that OpenStack per-

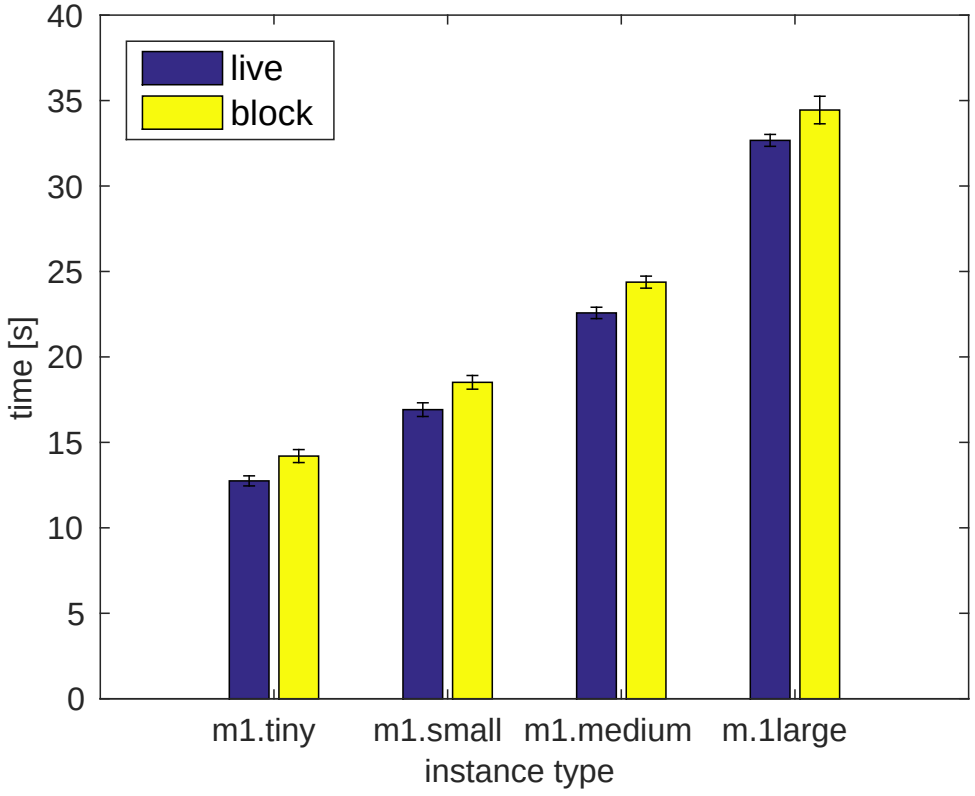

Figure 6.14: Migration duration for a throughput limit of 100 MBit/s with different instance flavors

forms considerable optimizations for the empty space not occupied by the booted images (see explanation at the beginning of Chapter 6).

As with the throughput limit the different parts of the migration do not scale in the same way with the instance flavor. Again the preoperation time provides the largest part of the duration and also scales the most. Figure 6.3 shows distinguishable increases of the preoperation time with the increase in instance size. The grows always exceed the confidence intervals. The effect especially becomes apparent at lower speeds. At 10 MBit/s the migration of a large instance takes about five times as long as the migration of a tiny instance. Like already detailed in 6.2.1 the live migration is always superior to the block migration.

For the downtime no global conclusions can be drawn without reasonable doubt. As seen in Figure 6.4 in most cases the confidence intervals overlap for the measurements have large overlapping section often continuing the average value. This is particularly the fact for migrations of the same type. So only conclusions for edge cases like "at 100 MBit/s live migration for the tiny flavor is always faster than block migration for the large flavor" can be drawn. These edge cases have only none to little significance for scientific evaluation as well as for practical applications. Therefore the effect of the instance flavor on the downtime can be considered negligible.

Figure 6.5 shows that the postoperation times are very close together. As detailed for the throughput limit in Section 6.2.1 no significant influence of the instance flavor can be shown beyond reasonable doubt. The same goes for the migration mode.

Fixed Network Delay

Table 6.2 used in Subsection 6.2.2 also shows the measured duration, preoperation, postoperation and downtimes for different flavors for every throughput setting. In Section A the values from Table 6.2 can be found again sorted by latency instead of flavor in Table A.2.

Figure 6.15 shows how the migration durations change when the network delay is kept constant (in this example 50 ms) and the flavor changes. The instance size has a slight influence on the duration but the effect is much smaller than in case of the throughput limitation. This is related to the fact that the instance flavor mainly influences the amount of main memory and disk transferred (block) or synchronized by the migration and to a lesser extend the number of connections and requests that are used in the migration. The effect is therefore clearly below either CPU count, memory or disk space increase between the flavors. The instance flavor has no effect on the comparison of migration types. At any latency level (except zero ms) the live migration is slower than the block migration.

Since the preoperation time is again the largest part of the duration it is also the most affected by the different flavors as seen in Figure 6.7. The effect is slightly stronger than for the total duration but still nowhere near the increase of CPUs, main memory or disk storage. The confidence intervals between two flavors often overlap but the increase to the second to next flavor usually corrects that problem. Again the competition between live and post migration is not influenced by the

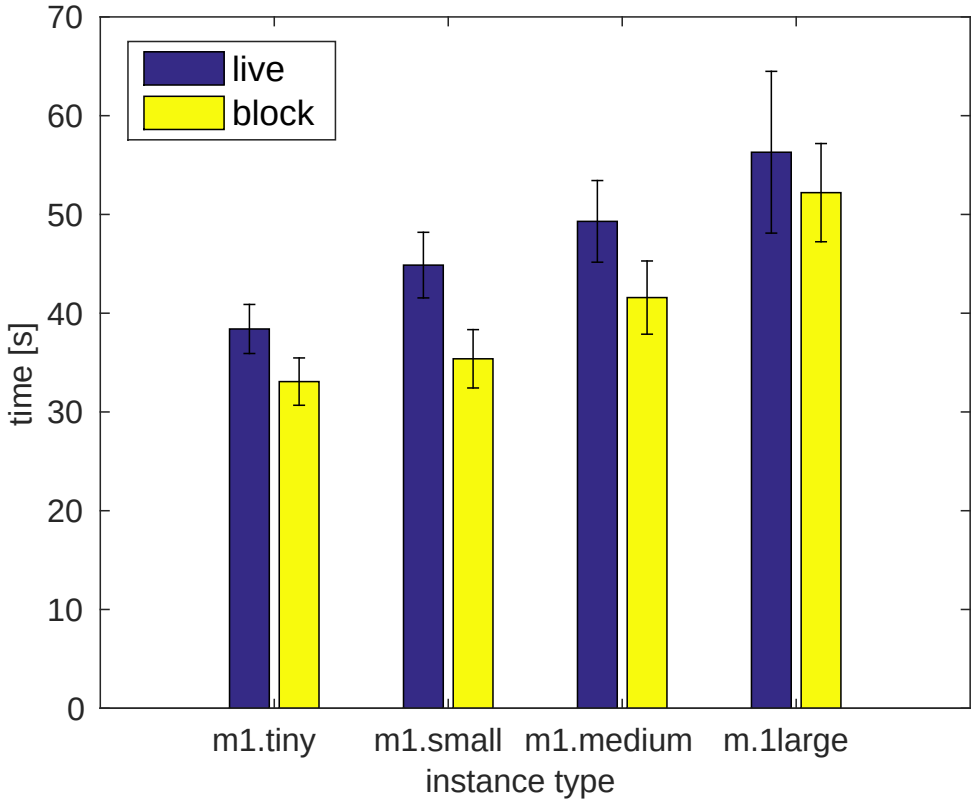

Figure 6.15: Migration duration for a network latency of 50 ms with different instance flavors

flavor. For every flavor the block migration is faster with latency above zero. Only when no latency is added the live migration is superior for all flavors.

As shown in Figure 6.8 the downtime shows no distinguishable difference between flavors with the exception of the tiny flavor with block migration at 50 ms delay. Its downtime exceeds all other downtimes which is surprising since it would be expected that the tiny flavor would have the fastest migrations. This is possibly explainable by OpenStack changing the migration optimization for the tiny flavor earlier than for the other flavors (the other flavors perform the same jump in downtime at 100 ms). Disregarding this singular exception the migration times are very similar between all flavor with no distinct difference provable beyond reasonable doubt. Also the flavor has no influence on the superiority of the live migration that has shorter downtimes for all flavors and latencies.

As can be seen in Figure 6.9 the postoperation time is not significantly influenced by the instance flavor. This is logical since in the postoperation phase most of the data transfer is completed and the main part is the reattachment of the network. These operations are not dependant of the network. Neither live nor block migrations have significant advantages in this area and this does not change when varying the instance flavor.

Fixed Loss Rate

Table 6.3 used in Subsection 6.2.3 also shows the measured duration, preoperation, postoperation and downtimes for different flavors for every throughput setting. In Section A the values from Table 6.3 can be found again sorted by latency instead of flavor in Table A.3.

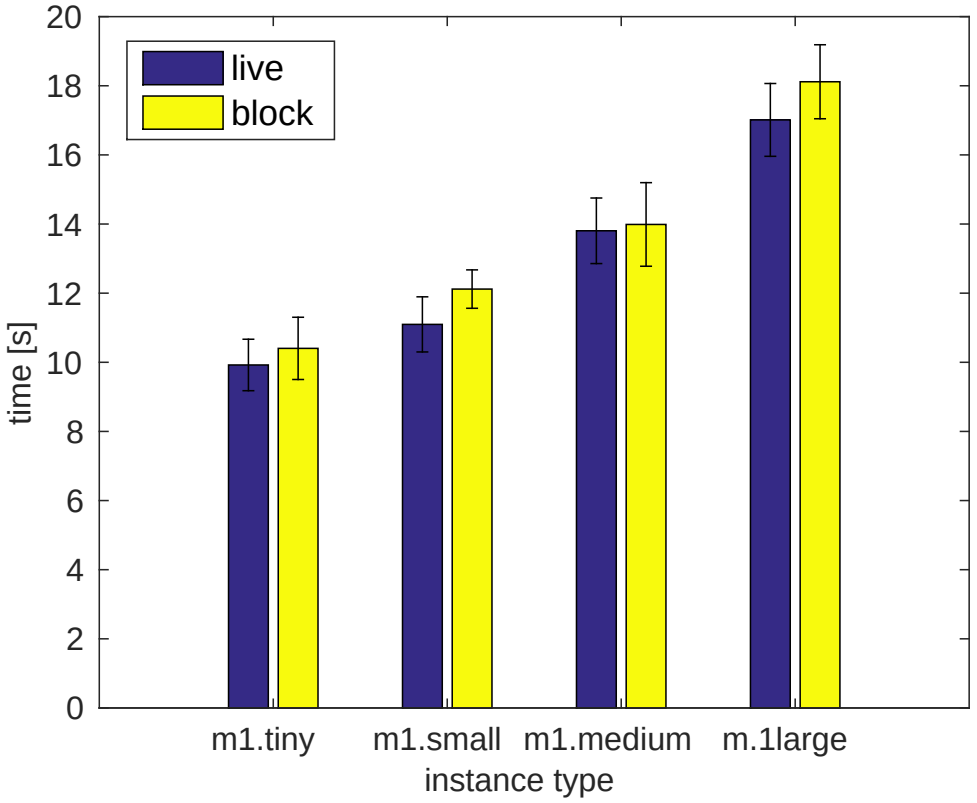

Figure 6.16: Migration duration for a network drop rate of one percent with different instance flavors

Figure 6.16 shows the migration duration of different flavors for a network loss rate of one percent. There is a slight but significant increase when switching to larger flavors. Again the effect is much smaller than the increase of the number of CPUs, the main memory or the disk space. While this is true for one percent the factor increases for five percent drop rate where the increase is nearly at the factor at which the main memory increases. The drop rate influences small connections as well as the transfer of large files (e.g. the snapshots and the file system synchronization). Therefore since larger flavors require more files to be transferred these operations are directly affected by the loss rate. The different flavors have no influence on the competition of live and block migration with the duration for the live migrations always being smaller than for the block migrations.

The largest part of the duration as well as the increase is as for the other factors the preoperation time as seen in Figure 6.11. Its increase is slightly higher than that

of the total duration. The increase is already significant between two flavors since the configuration interval do not overlap. Live migration is faster than the block migration for any flavor over all drop rates.

As shown in Figure 6.12 there is no significant influence of the flavor on the downtime with the exception of the live migration of the small factor at the one percent drop rate. There is no visible explanation of this exception. The only possibility could be that the small flavor has a anomaly optimal packet distribution for this scenario. A detailed understanding of this result would require a deeper analysis of the migration process as well as the transferred content. In the drop rate scenario there is no clear advantage of either live or block migration which is not changed by the varying instance flavors. The confidence intervals are very large due to the highly varying measurement results and therefore do not allow drawing safe conclusions.

Figure 6.13 depicts the duration of the postoperation phase for different flavors at different drop rates. IT is visible that there is no significant effect between the flavors. Also there is no significant advantage of either migration mode for any flavor.

6.2.5 Summary and Conclusions

The influence of different network characteristics on the migration has been analyzed. This was done for limitations of the network throughput as well as for delays on the network link and an increased chance of packet loss.

For the throughput limit it has been found, that the effect is significant for moderate network speeds (10 MBit/s) but not as strong for differences in the higher areas (100 MBit/s vs. 1000 MBit/s). The effect originates nearly completely from the preoperation time and slightly from the downtime while the postoperation time is not affected. The live migration is always faster than the block migration in the disciplines of total duration, preoperation time and downtime while both modes tie at the postoperation time.

Increasing the latency also increases the migration duration in a nearly linear manner over all latency levels. Here, the largest part is again the preoperation time that is strongly affected by the increase. The downtime is slightly increased. Contrary to the throughput limitation measurements the postoperation time is also affected in a nearly linear fashion. The live migration loses its advantage in the duration and preoperation time disciplines as soon as the latency is increased and is overtaken by the block migration. The live migration is still faster when the downtime metric is used while both modes are equally fast or slow for the postoperation time.

If the network reliability suffers and the rate at which packets are dropped increases, the total duration is significantly increased. Again most of the duration as well as its increase is credited to the preoperation time. The downtime is increased by a small factor. The postoperation time is growing by an even smaller factor. The live migration is superior when it comes to the total duration and the preoperation time. The high variations introduced by the higher drop rate prohibit the distinction of a superior mode for the downtime. The postoperation time is again not influenced by the migration mode.

Larger instance flavors increase the duration of a migration as well as the preoperation time for all modes and all the three previous measurements. The magnitude of the effect depends on the circumstances provides by the previous setups. Since network throughput and drop rate directly relate to the slower transmission of larger files across the network, the flavors have the largest effect when either a bandwidth limit is introduced or packets are dropped while the effect of the flavors in the scenario of increased latency is still significant but relatively small. Optimizations done by OpenStack allow to reduce the effect of the increased disk space of the virtual machines if that space is not entirely used. For the factors modifiable on the cloud's controller node namely the flavor and the migration mode recommendations can be given. Except for high latency networks the use of the live migration almost always guarantees faster migration times, preoperation times and especially downtimes. For the flavor the suggested guideline should be "As big as necessary, as small as possible." to minimize VCPU and RAM utilization while at the same time reducing the time spent on migrating.

6.3 Evaluation of Migration Performance from Application Perspective

Applications have different requirements regarding the migration process. There are less time critical applications like downloads of large files where most users would not notice a short interruption of the file transfer. However there are also more time critical application with different requirements. Normal video streaming requires an acceptable quality with no stalling while live streaming requires a short transmission time to user as well. Most critical are game servers running real time games where a lapse of a few milliseconds can decide between the players victory or defeat.

The usability of some applications during migration has been tested to evaluate the effect of the migration on the application and ultimately the QoE. The applications have been selected to cover multi and single user applications as well as stateful and stateless scenarios.

If not differently stated all applications are tested in two scenarios. The first scenario features no extra latency and a throughput of 1000 MBit/s is similar to the migration inside a data center. The migration at 100 MBit/s and 100 ms delay resembles the migration over a link between remote data centers.

6.3.1 Assessment for File Downloads

For the first evaluation if any application relying on networking conditions can be migrated a less time critical application was measured. The choice fell on a typical download. The UNIX tool GNU Wget[30] was chosen. Wget is a simple command line downloading tool. It is does not use optimization approaches like downloading multiple parts of a file at once or downloading from multiple servers. The download was started on the client by the command `wget --report-speed=bits --append-output=/path/to/logfile <URL>`. On the server side the nginx[31] has been chosen which is very common in UNIX server environments and aims at re-

placing the wide spread Apache web server. The network speed to the download client was limited to 100 MBit/s to prevent data transferred during the migration from influencing the download speed since they share a physical link at the compute node. For the download, a file of 10 GB size was chosen filled with random numbers to ensure no optimization or compression takes place. The virtual machine was instantiated with an Ubuntu Linux image and the m1.small flavor. The instances were migrated using the live migration mode.

During the running download the migration was initiated. The download speed was logged by the Wget command and written into a file. This was done for multiple latencies since as seen in Section 6.2 the latency has the most influence on downtime and postoperation time which are the most important characteristics for the compute and network operation.

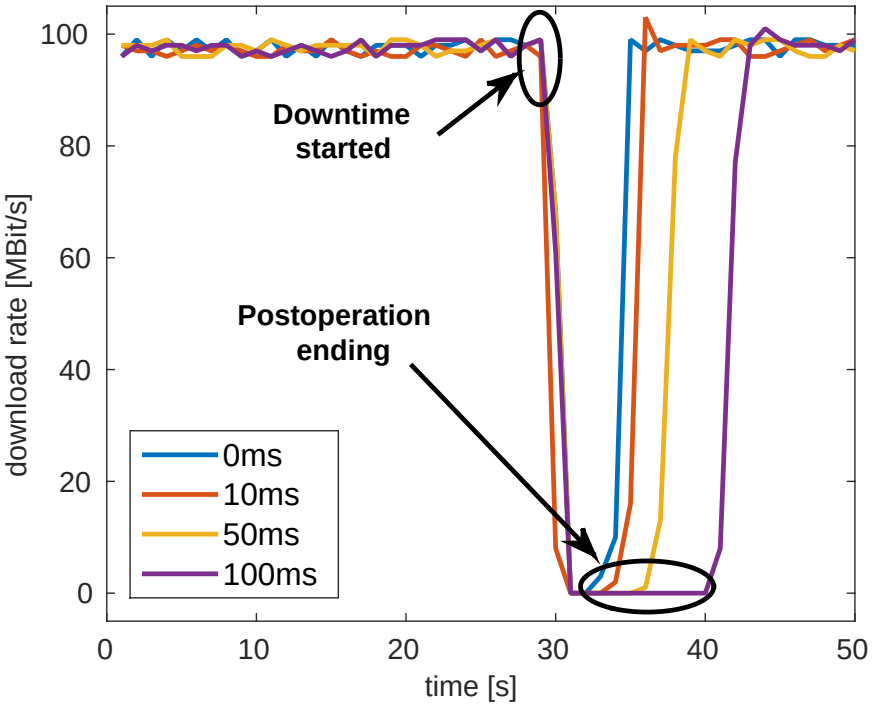

Figure 6.17: Download speed over time for download processes running while migrations are performed.

Figure 6.17 shows the development of the download speed over time. The different measurements have been aligned so that the migration always starts at thirty seconds. A this point in time the speed falls to zero. Since the speed is measured at a precision of one second the second the transmission is lost has no zero value. After downtime and postoperation time are passed the download resumes. The time frames at which the speed is at zero MBit/s correlate to the expected sum of downtime and postoperation time from Table 6.2. The maximum network speed is again reached after a few moments. The speed of the convergence is related to the use of Linux's CUBIC TCP [32] that allows a very fast adaption to the network speed

after packet losses.

The download was able to resume in any case. Even though the web server was guarded by a watch dog to resume it in case of a crash that was not necessary. The download application can therefore be considered fully migratable.

6.3.2 Server-based Streaming Using Content in the Virtual Machine

A more critical application is video streaming, especially live video streaming. The latter requires the parallel encoding and distribution of the material. If both tasks are to be performed one a single machine this puts high requirements on the CPU to achieve the encoding in real time. For this application, the widely utilized ffmpeg [33] a complete, cross-platform solution to record, convert and stream audio and video was used. For the encoding, the actual ffmpeg application was used encoding the source video into a buffer file for the streaming server. For the streaming itself ffserver was used streaming from the aforementioned buffer file. As video the Sintel [34] movie was chosen. The choice fell due to the free accessibility of the movie so the results can easily be verified as well as due to the popularity of this movie in demonstration in computer science publications. In this scenario, the streaming intelligence is supposed to be on the server side. Therefore for playback the tool MPlayer[35] has been chosen. MPlayer is an older but reliable and still maintained player software. MPlayer does not automatically reconnect or adapt the streaming quality but just plays the video from the provided URL. A cache can be set to compensate for variations in the transmission speed.

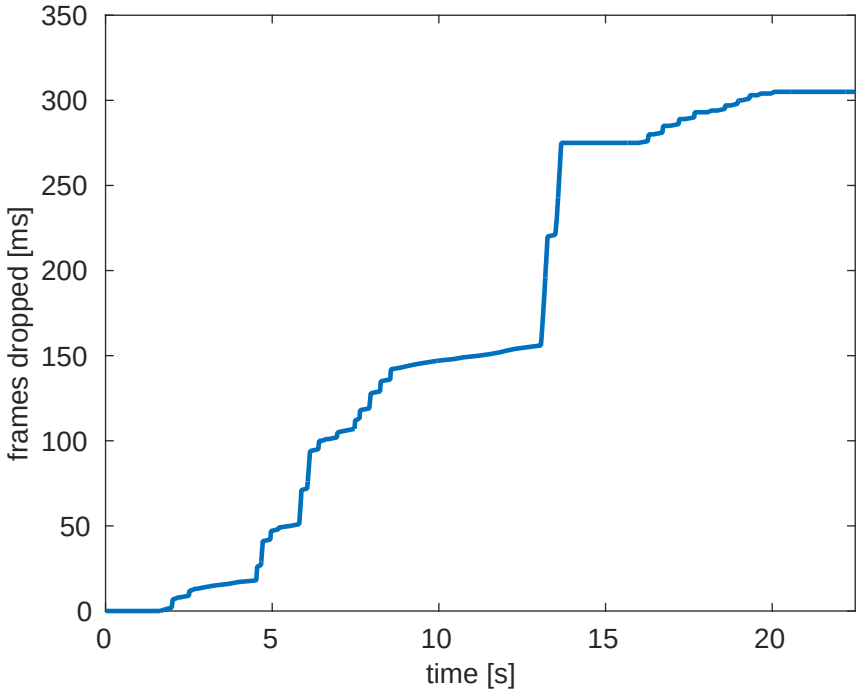

Figure 6.18: Frames dropped during the beginning of the migration process

To evaluate the migratability of this scenario a server of the m1. large flavor has been setup up. The large flavor was necessary to provide enough computing power to run the encoding since it provides four VCPUs.

After the migration is triggered some transmission errors occur when no cache is enabled. These lead to a large number of frames dropped as seen in Figure 6.18. To the user this is visible by stuttering of the stream and sudden jumps a few seconds ahead in the video playback. If sufficient caching is enabled this problem is not visible to the user but the filling level of the cache drops slightly. After most of the machine is migrated OpenStack continues the server migration taking a lot of time to transfer the larger part of the snapshot. The log file shows that the amount of content to transfer permanently increases. This is due to the encoder permanently encoding the video and writing to the buffer file. This effect is that serious strong that the migration process is getting stuck permanently alternating between zero and five percent remaining but can not complete the migration even after 20 minutes. The moment the encoding is remotely terminated it takes but a few seconds to finish the migration.

The experiment has been repeated with the settings for the inter data center migration scenario. The only change was that it took longer to reach the state around zero percent. Then the migration also stucked.

The fact that the migration not only noticeably impairs the playback quality but in fact the migration can not be completed renders this application scenario not migratable.

6.3.3 Server-based Streaming of External Content

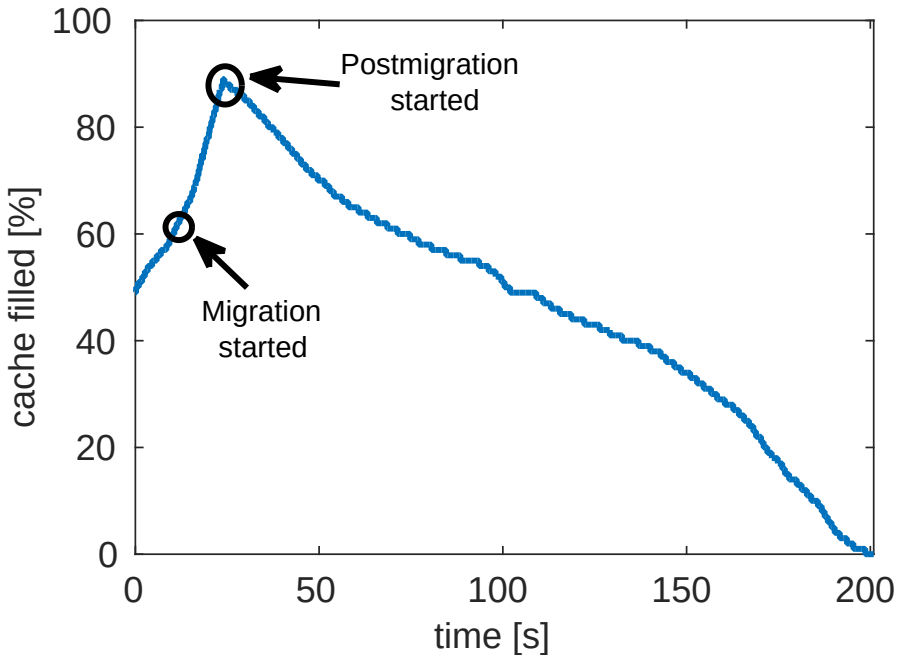

Figure 6.19: Percentage of buffer fill level over time before, during and after migration

Since the concept detailed in Subsection 6.3.2 fails the demand of migratability the idea is to move the encoding to a separate node and migrate only the node that is running the streaming server. On the new dedicated content server again ffmpeg was used for encoding. The streaming was again done using ffserver inside the virtual machine. The Sintel video was used as well as the MPlayer on the client. Since the amount of processing power required was significantly reduced, it was possible to switch to the m1.small flavor.

In the intra compute center scenario the migration is always performed - meaning the virtual machine is moved to the other server and the ffserver Unix process does not crash. Unfortunately in 48% of the migration runs the client disconnects during the postoperation phase. If the cache is enabled, it plays till the cache is depleted as seen in Figure 6.19. If no cache is enabled, it instantly crashes. Resuming the stream is possible but needs the manual intervention of restarting the player program. Therefore this scenario is only semi-migratable with a high chance of failure making it inapplicable for productive use.

A very interesting behaviour occurs when migrating in the inter compute center scenario with increased network delay. One might usually expect that under worse condition the QoE during the migration would further decrease. This is not the fact. The total migration takes longer especially due to the increase latency. The machine is successfully migrated and the process continues to run. The connection stays stable with the cache not filling for a short amount of time that correlates

with the lengths of downtime and postoperation time from Table 6.2. However if no cache is selected the videos stalls that long and then continues where it was suspended. This is a very surprising behaviour. The only possible explanation is that the disconnect is not detected fast enough due to the already existing network delays. In this scenario the migration is fully migratable.

6.3.4 Dynamic Adaptive Streaming over HTTP (DASH)

As a final scenario for video streaming, a client-based streaming application was chosen. On the server side the material was provided in multiple quality levels by a web server and a playlist file was provided to the client. No further intelligence or optimization happened on the server side.

As a client the TAPAS (Tool for rApid Prototyping of Adaptive Streaming algorithms) player was chosen using the MPEG DASH streaming protocol which is a popular approach to video streaming. It implements an adaptive streaming controller as described in [36] meaning it is capable of adapting the chosen quality to environment parameters like the network speed and the current cache level. The client can chose between six different quality levels between 360 pixels vertical resolution and 2160 pixels vertical resolution (commonly known as 4K).

Obviously, providing the video in multiple quality levels and at once makes this scenario unattractive for the streaming of live transmissions like soccer matches. Nevertheless, it is a viable alternative for the streaming of preproduced material which makes out the larger part of the view content.

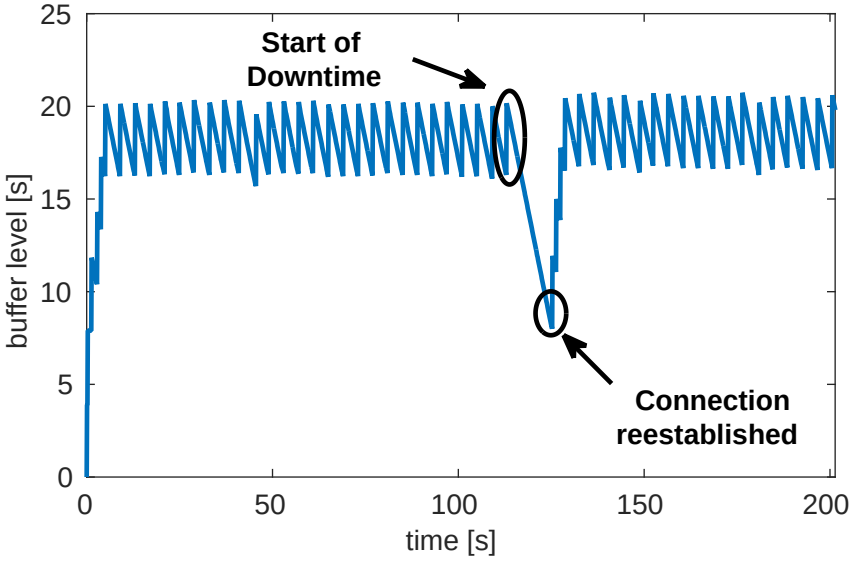

Figure 6.20: Buffer development over time during migration in an intra compute center scenario

In the first scenario with no speed limit or added latency depicting a migration inside a compute center the migration is successfully performed. The web server

application survives the migration without crashing and the client successfully reconnects and continues streaming the video. During the time where no network operation is possible the clients cache level drops. With the preset cache level of 20 seconds the migration is possible without endangering the fluid playback of the video stream as shown in Figure 6.20. If no cache or significantly less cache is configured the video hangs until the transmission is resumed. In the tested configuration after the first segment of the video the player constantly operates at the maximum quality possible even during the migration.

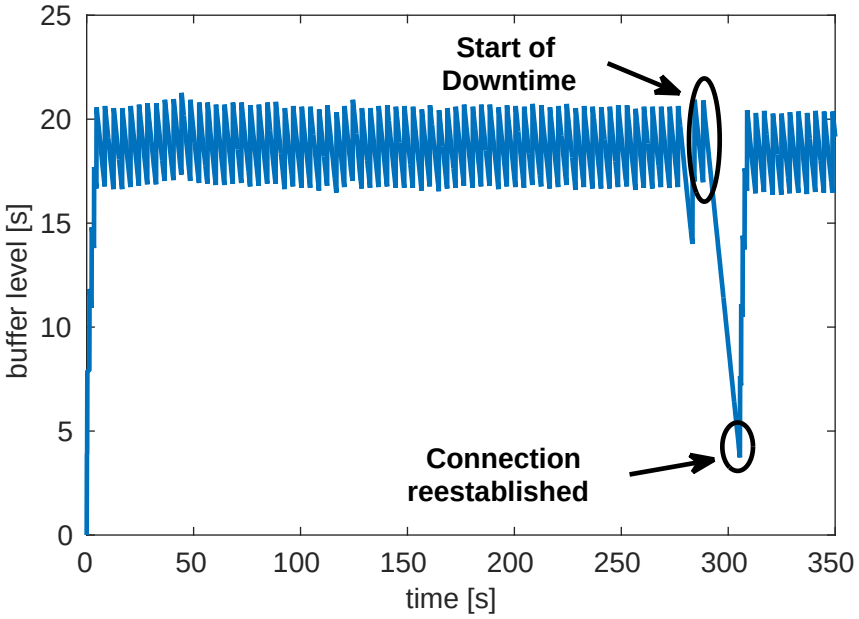

Figure 6.21: Buffer development over time during migration in an inter compute center scenario

The experiment was repeated for the inter compute center scenario. The first obvious change was the far prolonged migration duration as expectable from the results described in the preceding section. The migration was again performed successfully without any negative effects on the server application. Again the client reconnected successfully but this time the the buffer level came closer to zero as seen in Figure 6.21. With the configured cache level it was only just possible to prevent stalling. Also the server adapted due to the low buffer fill level and requested the first segment after reestablishment of the link at only the second best quality level to faster fill the cache. This was the only occurrence except for the ramp up when beginning playback where the quality was not at maximum level.

In conclusion the migrations for the client side streaming application were always successful with no failures or total loss of connection like the server side applications. This renders this solution complete migratable in the intra compute center scenario as well as in the inter compute center scenario making it the only video streaming scenario succeeding in both cases.

6.3.5 Counter-Strike: Source Online Gaming

As a classical scenario for a multi user application that has to store the internal state a game server has been chosen. The selected game was Counter-Strike: Source[37]. Counter-Strike: Source is a so called first person shooter, a very popular game type over two decades [38]. The game is played from the viewpoint of the selected character which is moved via keyboard input and weapons and tools are aimed using the mouse. The game was selected due to the excellent Linux support for client as well as server, simple server setup and because an activated license was already available.

The players are divided in two groups: The terrorists and the counter terrorists. Depending on the selected type of gameplay the goal varies. There are four basic modes. In the bomb defusal scenario the terrorists are tasked to either blow up one of two possible locations by placing a bomb and preventing the counter terrorists from defusing it or by eliminating the other team. The counter terrorists have to either eliminate the terrorist team before the bomb is placed or defuse the bomb before it explodes. In the hostage rescue scenario, the counter terrorists have to rescue hostages or alternatively eliminate the terrorist team while the terrorists must prevent the counter terrorists from rescuing all hostages or eliminate the opposite team. In the VIP escort scenario, the counter terrorists have to escort one special player to a rescue point. They win if they either reach that point or eliminate the opposite team while the terrorists win once the VIP is killed. In the final deathmatch mode the team wins which eliminates the other team. To achieve this goals the players may equip themselves with different types of weapons, armor and tools.

In every mode, winning requires fast reaction paired with high precision when aiming the weapons. In many cases a few milliseconds of reaction time can be a matter of live or death. Therefore during the first introduction of ADSL due to the higher latencies many professional and semi-professional gamers stayed with the low latency ISDN technology until the introduction of FastPath[39] made DSL competitive for that target group. Thus the target group is very critical when it comes to lags and stuttering. Migrations would have to be performed without even being noticed.

The dedicated game server was provisioned in a virtual machine of the m1.medium flavor installed with an Ubuntu 64bit image. The migrations have been randomly started during the gameplay so they occurred at different situations.

One of the first results was the realization that block migrations were not possible since they were aborted after some time because the downtime would exceed the internal limit set by OpenStack. Before the decision to abort, multiple seconds of heavy lag are perceivable making the game unplayable. Especially when players controlled by artificial intelligence (so called bots) were active on the server that did not suffer from the latency issues the human players stood no chance because of the severe lags. This rendered the block migration unusable for a real-time gaming scenario since not only the goal of moving the machine to another server was missed but also the game was unplayable while trying to do so.

Going on with the live migration the first test showed that the migration was performed successfully. The migrations took about 60 seconds at 1000 MBit/s with

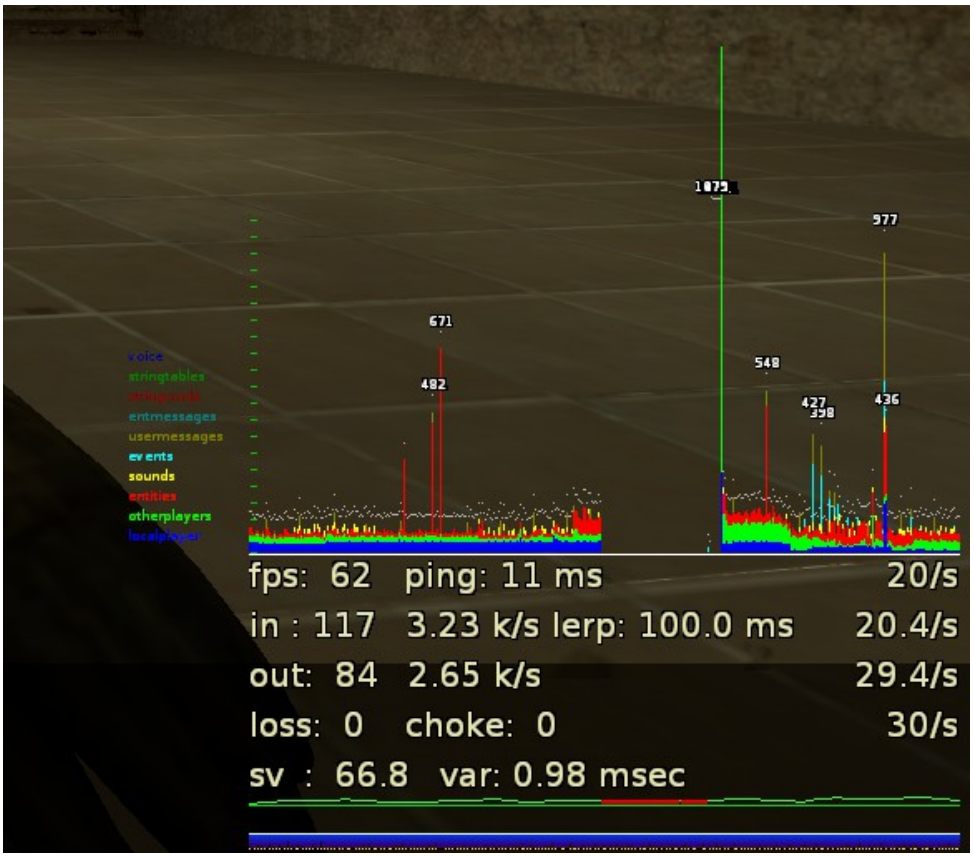

Figure 6.22: The netgraph shown by Counter-Strike: Source depicting the loss of data transmission during the migration's downtime and postoperation phase

no extra latency added, which is much more than expected from the measurements in the previous section. This can be explained to the fact, that the virtual machines hard drive consumption is larger than in the previous section since the game server had to be installed in the machine requiring about 2 Gigabytes of extra space. Additionally due to the continuing game files on the disk as well as the main memory where modified during the migration process requiring additional synchronization iterations.

The effect on the gameplay actually depends on the situation. If the migration entered the downtime and postoperation phase during a situation where the teams were far apart the games interpolation algorithm continued the movement and the players did not realize a migration has been performed. This fact was only visible if the game's internal netgraph was enabled as seen in Figure 6.22. Therefore in such situation the migration had no effect on the players' QoE. But if the downtime and postoperation phase were entered during a combat situation the synchronization between server and client led to stuttering and randomly moving the player to the position he was a few moments ago. Under these circumstances, it was impossible to even play somewhat effectively. Like for the stuttering of the block migration, potential bots had a huge advantage since they only were stuck during the downtime

but were already active during the postoperation time while human players were still waiting for the network to be reattached to the virtual machine.

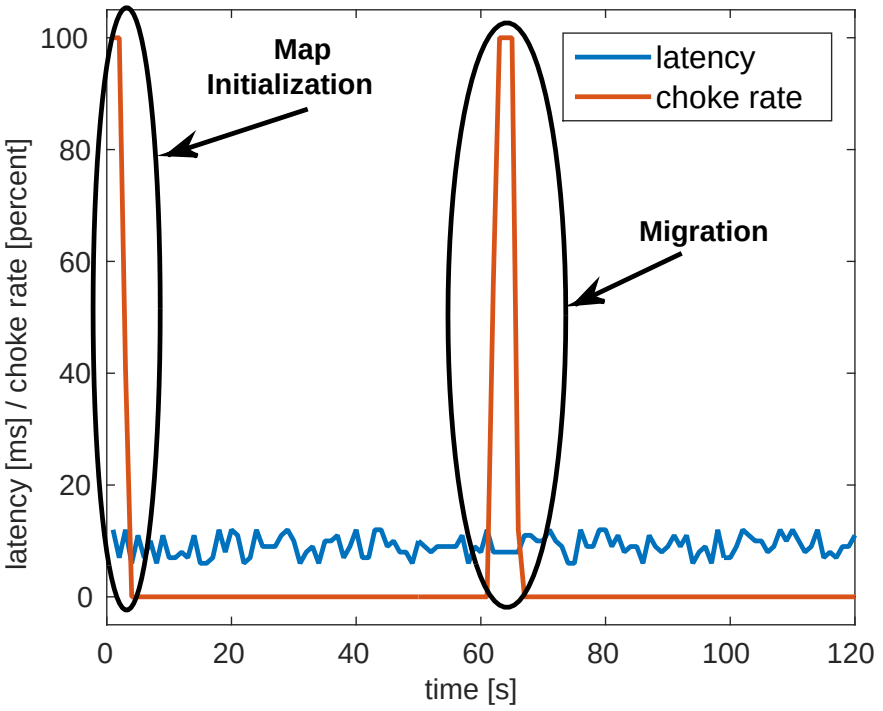

Figure 6.23: Latency and choke rate plotted over the first 120 seconds of a Counter-Strike: Source Match with a migration happening shortly after 60 seconds.

Figure 6.23 shows the development of the latency and the choke rate over the first 120 seconds of a Counter-Strike: Source match. The choke rate is the percentage of packages expected from the server that were not received. The game always chokes at the beginning of every match during the phase where the server reinitializes the map but this has no noticeable effect on the gameplay since the players' movement is prohibited for the first few seconds to allow for them buying equipment. The latency in our scenario oscillates around 11 ms. When the migration occurs the choke rate increases to 100 percent. The first and last second do not reach 100 percent since in those seconds a part of the data comes through. The time frame of about three seconds correlates to the estimated sum of downtime and postoperation time and a short time span for the client to realize the connection is up again.

The tests were also conducted for lower bandwidths resulting only in longer preoperation times while the actual downtime and postoperation time stayed about the same as shown in the previous sections. When migrating with higher latencies in the inter data center scenario, the time increases similar to the increase of the sum of downtime and postoperation time from Table 6.2. This leads to the delays being more recognizable to the player as well as the server automatically kicking the player for high latency after the third migration, rendering this scenario unmi-

gratable. Since different networks are used for the migration and the connection between the game server and the client the network transfer of the migrations had no influence on the players link quality.

Concluding the assessment of this application in the intra data center scenario it can be stated that the effect on the QoE is acceptable in some situations in the game but absolutely an unreasonable handicap for the users in others. As long as there is no way to exactly define the moment, the downtime and postoperation occur the only scenario where a migration would be acceptable is when the latency of the link between client and server is already so bad, that a short downtime would not make it any worse and afterwards the connection to the target server would be much better. The application is therefore at best semi-migratable in this scenario.

6.3.6 Summary and Overview of the Results

Application	Scenario 1	Scenario 2
Download	✓	✓
Server Side Streaming + Content	✗	✗
Server Side Streaming w/o Content	✗*	✓
Client Side Streaming	✓	✓
Video Gaming	✗*	✗

Table 6.5: Summary of the migratability of different applications. Where ✓means the migration is possible and works reliable and ✗means that it does not. ✗* means that the migration is successful some times but fails at others or causes non tolerable impairments. Scenario 1 resembles intra compute center migrations while Scenario 2 resembles inter compute center migrations.

In this section multiple applications have been analyzed for their migratability as shown in Table 6.5. As expected the more complicated the applications are and the more they rely on short reaction times they become harder to migrate without impairments or at all. While some applications become unusable others can not at all migrated. This shows that the simple migration approach is only usable for a small set of applications.

6.4 Application Aware Optimizations for the Migration Process

In the previous section, we have tested various applications for their migratability. It has been found that many of these applications can either not be migrated in their current state or they suffer such heavy impairments that they become unusable due to the migrations. To fix this shortcomings, it is necessary to optimize the process of migration with respect to the application state or find other solutions to ease the effect of the migration.

6.4.1 Optimizations for File Transfers

As seen before the download task is already migratable with only a short stop of the transmission. If for some reason even this drop needs to be minimized a load balancer must be added switching the download to backup server before the source server's downtime starts. If required the download could be switched back to the original server afterwards. For this the state of the download must be synchronized between the servers.

6.4.2 Server-based Video Streaming of Internal Content

The main problem in this scenario was on one hand the initial frame drop when starting the migration process. This could be generally resolved by reducing the migrations priority level so that no resources are taken away from the streaming process.

To circumvent the problem of the migration not taking place at all but stalling at a few percent remaining requires actual awareness of the application to still allow the migration without losing quality for the user. The key to the successful continuation of the streams transmission after the migration is that the buffer is not depleted. This is currently ensured by the encoding processes but exactly this process is the one keeping the migration from completing. Therefore this process must be halted for the migration to complete and resume afterwards. The time needed for the migration to complete the downtime and postoperation phase can be estimated from the tables in Section 6.2. The buffer must be at least filled with enough frames to cover that time span. In some scenarios this could mean increasing the buffer or switching to a lower quality setting to fill the buffer fast enough. A special problem occurs in the case of live streaming where the streaming server might be streaming an event only moments after it is happening in real life. To compensate for this an artificial delay must be introduced. This could be done during an advertisement block. After the migration this delay becomes unnecessary or can even cause bad QoE (e.g: During the soccer world cup in 2014 the ZDF live stream of the game Germany vs. Brazil led to many complaints since due to a sub optimal implementation the stream was up to two minutes behind the television signal.). Thus the added delay could be removed in another advertisement block.

6.4.3 Server-based Video Streaming of External Content

The main weakness of this scenario was that almost half of the time the client lost its connection to the server and required manual reconnecting. This could be solved by adding additional intelligence to the client telling it to automatically reconnect. The time frame for the reconnection could be estimated by taking the tables from Section 6.2. In the best case there would be a protocol where the server would tell the client that it is soon going down allowing the client to increase its cache level if possible and then telling the client that the downtime begins now and in how many seconds the resume is expected. Being aware of the surrounding network characteristics the server could already tell the client at the beginning that migrations are possible and

the client should adjust its cache size accordingly to the expected downtime and postoperation time.

6.4.4 Dynamic Adaptive Streaming over HTTP (DASH)

As shown before this approach is already migratable without significant losses in quality. As written about the server based streaming by increasing the clients intelligence and adding information about the estimated migration factors as well as currently scheduled migrations could increase the quality even further. This can be optimized to an amount where the user is guaranteed not to realize the migration taking place.

6.4.5 Counter-Strike: Source Online Gaming

As written before, the user acceptance of the migration is very dependant on the situation in which it takes place. In the case of Counter-Strike: Source, migrations during the spawn, buy and initial phase where not at all visible to the players. Therefore a migration should be performed in this phase. A first approach with only information from the game would be to trigger a migration so that the estimated time for the migration phases where no network transmission is available falls on the tolerant phases of the game. In the case of Counter-Strike: Source such an estimation is nearly impossible since games can take anywhere between over five minutes and under thirty seconds. There are games that are more predictable. For these that approach would be practical. For unpredictable games a useful approach must not only predict when the migration should happen but must be able to control when it happens. Therefore an application aware module would have to be introduced into OpenStack performing the complete preoperation phase of the migration and then halt the migration (only continuously updating the deltas) until a situation appears where a short loss of transmission is acceptable and the trigger the switch to the downtime and postoperation phase. With such an approach it would be possible to perform migrations in the intra data center scenario without the players noticing. A successful migration in the inter data center scenario is not possible even with this optimization since the duration of the non responsive migration phases is too long and would reach into the critical phases of the game.

6.4.6 Summary and Lessons Learned

In this section approaches to improve the migratability of different applications have been shown. The approaches vary depending on the application so no general all-purpose solution can be given. The improvements would require more intelligence from the applications themselves making them aware of them being migrated and then warning the clients about that or even modifications of the cloud software to control the migration process based on the state of the used application.

7 Conclusion and Outlook

This chapter concludes the thesis and presents the most important findings. Due to the increasing popularity of cloud services, resource management with respect to the users' perceived quality, as well as in terms of energy efficiency and cost, is becoming more and more important in a cloud infrastructure. A fundamental part within this resource management process in the cloud is the migration of cloud services. The decision whether the benefits of a migration justify the migration effort requires addition information and studies. This includes in particular the influence of the migration on the actual service performance, the duration of the migration, and influences of different network parameters on these two factors.

In this thesis at first a literature research on the topics of cloud services, resource allocation, orchestration and migration has been performed. Additionally modeling for cloud services and the placement of resources has been considered.

A testbed for the measurement of cloud migrations has been defined and realized. It features a basic OpenStack cloud environment with a central control node as well as two dedicated compute nodes. The testbed is provided using automatic orchestration with the software tool puppet allowing for the fast addition of more compute nodes or other OpenStack services. Additionally, a log management solution based on Logstash, Redis and ElasticSearch has been introduced to collect as much information as possible about the cloud environment. The solution is a scalable architecture making it applicable for larger systems with hundreds of nodes. The overhead of the logging has been proven to be negligible for the used setup as well as for larger systems. The system features virtual networks splitting the traffic of the different services to allow the independent modification of the network parameters for each virtual link. An automatic benchmarking algorithm has been implemented allowing for a preselect number of migrations to be performed with configured network parameters. It allows for batch processing, running multiple different measurements consecutively.

Using this testbed we measured and evaluated the impact of network characteristics as well as different virtual machine instance flavors on the migration performance. Reduced bandwidth, added latency and increased packet loss rate have been evaluated for their influence. While all of these increase the total duration significantly, the influence of latency and drop rate on the downtime is more severe than the influence of the bandwidth limitation. The postoperation time is only increased significantly when increasing the network latency. The most significant increase in total migration duration was generated by the loss rate. For the loss rate and the bandwidth limitation scenarios the live migration proved to have the shorter migration period while with limited throughput also providing the shorter downtime. In the delay scenario the scales are tipped making the live migration the slower migration while still having the shorter downtime. Larger instance flavors led to longer

migration duration. The effect is more significant for the bandwidth limitation and drop rate experiments, that directly influence the network transmission speed, than for the latency scenario.

Next, the migration performance was analyzed from application perspective. To assess this performance multiple applications have been chosen and migrated inside the testbed while clients were connected and using the provided services. The assessed applications include file transfers, server-based video streaming with and without content inside the virtual machine, adaptive streaming using DASH, and online gaming. While the applications with little server side intelligence had few problems with the migrations the more complex applications failed to migrate at all or suffered severe impairments to the usability. More precisely the server side video streaming with included content did not migrate at all while on the other hand, the DASH streaming and the file transfer migrated without problems.

Based on the application evaluation of the migration performance, multiple optimizations using application awareness have been suggested that optimize the service quality when migrating. Additional intelligence is required either timing the migration based on an internal application state or telling the client or server to perform certain actions shortly before the migration is scheduled to occur.

The results give a detailed overview of the effects of network parameters on the migration performance. Additionally, it has been proven that migrations of complex applications still result in a strong impairment of the service quality.

Future work may deal with the realization of the proposed application-aware migration solutions. While this work focuses on the network parameters, the evaluation of the effect of other parameters like hard disc I/O load and memory utilization could yield interesting results. Leaving the path of deducting correlations from measurements but instead switch to an analytic approach, the modelling of the algorithms behind the different migration types would be required.

Acknowledgement

I would like to thank my thesis supervisor Florian Wamser for spending countless hours at night and at the weekend performing benchmarks together and discussing our further work.

I would also like to thank my girlfried Anja Razinskas for tolerating me not seeing her for multiple weekends to finish the thesis in time.

I want to thank my boss Prof. Samuel Kounev for being understanding when I once again completed a work asignment at the last possible minute due to being entangled in the thesis.

I finally want to thank my parents and my friends who supported me during this time and wished me luck and success.

This work was partialy supported by German Research Foundation (DFG) under Grant No. KO 3445/11-1. and the H2020 INPUT (Call H2020-ICT-2014-1, Grant No. 644672).

A Appendix

Additional Tables with Different Grouping

This section shows several table already existing in the main part of the thesis but with different sorting strategies to facilitate the comparison of different scenarios.

Mode	Flavor m1.	Speed [MBit/s]	avg. Duration [s]	avg. Preoptime [s]	avg. Downtime [s]	avg. Postoptime [s]
block	tiny	10	76.606 ± 5.324	74.580 ± 5.305	0.403 ± 0.075	1.623 ± 0.122
block	small	10	119.284 ± 3.917	117.307 ± 3.956	0.388 ± 0.055	1.588 ± 0.085
block	medium	10	172.856 ± 0.743	170.847 ± 0.778	0.393 ± 0.110	1.616 ± 0.121
block	large	10	271.278 ± 0.417	269.373 ± 0.375	0.352 ± 0.041	1.554 ± 0.137
block	tiny	100	14.200 ± 0.380	12.372 ± 0.354	0.188 ± 0.060	1.638 ± 0.123
block	small	100	18.513 ± 0.401	16.725 ± 0.374	0.181 ± 0.062	1.606 ± 0.115
block	medium	100	24.375 ± 0.352	22.571 ± 0.326	0.206 ± 0.066	1.598 ± 0.178
block	large	100	34.446 ± 0.806	32.656 ± 0.676	0.248 ± 0.078	1.541 ± 0.115
block	tiny	1000	12.333 ± 0.712	10.521 ± 0.694	0.214 ± 0.074	1.598 ± 0.127
block	small	1000	14.443 ± 0.965	12.628 ± 0.865	0.228 ± 0.089	1.587 ± 0.154
block	medium	1000	18.045 ± 1.004	16.138 ± 0.900	0.224 ± 0.074	1.683 ± 0.155
block	large	1000	23.579 ± 0.983	21.766 ± 0.863	0.189 ± 0.061	1.624 ± 0.166
live	tiny	10	65.628 ± 1.963	63.478 ± 5.305	0.390 ± 0.094	1.760 ± 0.105
live	small	10	107.791 ± 1.598	105.725 ± 3.956	0.414 ± 0.140	1.652 ± 0.128
live	medium	10	163.559 ± 0.647	161.413 ± 0.778	0.369 ± 0.061	1.777 ± 0.102
live	large	10	264.068 ± 3.818	261.961 ± 0.375	0.406 ± 0.080	1.702 ± 0.148
live	tiny	100	12.747 ± 0.295	10.940 ± 0.354	0.134 ± 0.014	1.673 ± 0.136
live	small	100	16.915 ± 0.401	15.149 ± 0.374	0.188 ± 0.100	1.578 ± 0.094
live	medium	100	22.575 ± 0.332	20.797 ± 0.326	0.222 ± 0.082	1.556 ± 0.102
live	large	100	32.670 ± 0.348	30.789 ± 0.676	0.228 ± 0.042	1.653 ± 0.121
live	tiny	1000	10.726 ± 0.423	8.967 ± 0.694	0.127 ± 0.023	1.630 ± 0.070
live	small	1000	13.658 ± 0.499	11.921 ± 0.865	0.107 ± 0.022	1.630 ± 0.057
live	medium	1000	16.393 ± 0.454	14.609 ± 0.900	0.140 ± 0.036	1.643 ± 0.069
live	large	1000	22.389 ± 0.381	20.566 ± 0.863	0.154 ± 0.015	1.668 ± 0.066

Table A.1: Migration durations sorted by throughput limits with ascending flavor with confidence intevals with 95% confidence level

Mode	Flavor m1.	Delay [MBit/s]	avg. Duration [s]	avg. Preoptime [s]	avg. Downtime [s]	avg. Postoptime [s]
block	tiny	0	7.531 ± 00.116	5.913 ± 00.107	0.253 ± 0.025	1.365 ± 0.054
block	small	0	8.047 ± 00.193	6.457 ± 00.191	0.280 ± 0.063	1.310 ± 0.084
block	medium	0	8.741 ± 00.182	7.075 ± 00.185	0.299 ± 0.034	1.367 ± 0.029
block	large	0	10.241 ± 00.274	8.594 ± 00.235	0.302 ± 0.049	1.345 ± 0.079
block	tiny	10	10.894 ± 00.321	8.581 ± 00.437	0.308 ± 0.067	2.005 ± 0.204
block	small	10	11.569 ± 00.558	9.654 ± 00.474	0.226 ± 0.078	1.688 ± 0.274
block	medium	10	12.910 ± 00.823	10.598 ± 00.886	0.332 ± 0.086	1.980 ± 0.202
block	large	10	14.342 ± 00.526	12.133 ± 00.577	0.321 ± 0.073	1.888 ± 0.241
block	tiny	50	33.074 ± 02.396	27.778 ± 02.921	1.101 ± 0.313	4.195 ± 1.125
block	small	50	35.383 ± 02.954	30.326 ± 03.238	0.505 ± 0.261	4.552 ± 0.754
block	medium	50	41.581 ± 03.707	36.347 ± 03.830	0.652 ± 0.272	4.583 ± 0.796
block	large	50	52.206 ± 04.972	46.664 ± 05.114	0.793 ± 0.247	4.750 ± 0.524
block	tiny	100	56.068 ± 02.590	45.924 ± 02.331	1.222 ± 0.768	8.922 ± 1.656
block	small	100	62.779 ± 03.741	53.576 ± 03.688	1.112 ± 0.664	8.091 ± 0.936
block	medium	100	73.490 ± 04.260	64.120 ± 04.004	1.426 ± 0.801	7.944 ± 1.322
block	large	100	86.144 ± 10.469	76.539 ± 10.288	1.420 ± 0.737	8.185 ± 1.392
live	tiny	0	6.829 ± 00.237	5.076 ± 00.228	0.271 ± 0.049	1.482 ± 0.055
live	small	0	7.524 ± 00.204	5.779 ± 00.213	0.260 ± 0.059	1.485 ± 0.058
live	medium	0	8.228 ± 00.209	6.415 ± 00.219	0.343 ± 0.062	1.469 ± 0.029
live	large	0	9.417 ± 00.207	7.686 ± 00.198	0.341 ± 0.051	1.390 ± 0.086
live	small	10	12.790 ± 00.613	10.216 ± 00.589	0.187 ± 0.092	2.386 ± 0.140
live	tiny	10	11.864 ± 00.423	9.322 ± 00.325	0.263 ± 0.159	2.279 ± 0.454
live	medium	10	13.618 ± 00.628	11.043 ± 00.596	0.203 ± 0.030	2.371 ± 0.278
live	large	10	16.069 ± 01.568	13.710 ± 01.662	0.191 ± 0.035	2.168 ± 0.365
live	tiny	50	38.402 ± 02.483	32.889 ± 02.496	0.372 ± 0.093	5.141 ± 1.163
live	small	50	44.869 ± 03.322	39.080 ± 03.111	0.430 ± 0.058	5.359 ± 0.961
live	medium	50	49.298 ± 04.133	42.721 ± 04.234	0.519 ± 0.241	6.059 ± 1.577
live	large	50	56.299 ± 08.190	50.608 ± 08.585	0.417 ± 0.117	5.274 ± 1.321
live	tiny	100	69.905 ± 07.106	60.089 ± 07.093	0.809 ± 0.515	9.007 ± 1.120
live	small	100	79.181 ± 04.538	69.265 ± 05.341	0.665 ± 0.071	9.251 ± 1.798
live	medium	100	84.653 ± 06.314	75.539 ± 06.050	0.677 ± 0.087	8.437 ± 1.615
live	large	100	96.000 ± 12.398	86.239 ± 11.989	0.585 ± 0.201	9.176 ± 1.411

Table A.2: Migration durations sorted latency with ascending flavor with confidence intevals with 95% confidence level

Mode	Flavor m1.	Delay [MBit/s]	avg. Duration [s]	avg. Preoptime [s]	avg. Downtime [s]	avg. Postoptime [s]
block	tiny	0	7.531 ± 00.116	5.913 ± 00.107	0.253 ± 0.025	1.365 ± 0.054
block	small	0	8.047 ± 00.193	6.457 ± 00.191	0.280 ± 0.063	1.310 ± 0.084
block	medium	0	8.741 ± 00.182	7.075 ± 00.185	0.299 ± 0.034	1.367 ± 0.029
block	large	0	10.241 ± 00.274	8.594 ± 00.235	0.302 ± 0.049	1.345 ± 0.079
block	tiny	10	10.894 ± 00.321	8.581 ± 00.437	0.308 ± 0.067	2.005 ± 0.204
block	small	10	11.569 ± 00.558	9.654 ± 00.474	0.226 ± 0.078	1.688 ± 0.274
block	medium	10	12.910 ± 00.823	10.598 ± 00.886	0.332 ± 0.086	1.980 ± 0.202
block	large	10	14.342 ± 00.526	12.133 ± 00.577	0.321 ± 0.073	1.888 ± 0.241
block	tiny	50	33.074 ± 02.396	27.778 ± 02.921	1.101 ± 0.313	4.195 ± 1.125
block	small	50	35.383 ± 02.954	30.326 ± 03.238	0.505 ± 0.261	4.552 ± 0.754
block	medium	50	41.581 ± 03.707	36.347 ± 03.830	0.652 ± 0.272	4.583 ± 0.796
block	large	50	52.206 ± 04.972	46.664 ± 05.114	0.793 ± 0.247	4.750 ± 0.524
block	tiny	100	56.068 ± 02.590	45.924 ± 02.331	1.222 ± 0.768	8.922 ± 1.656
block	small	100	62.779 ± 03.741	53.576 ± 03.688	1.112 ± 0.664	8.091 ± 0.936
block	medium	100	73.490 ± 04.260	64.120 ± 04.004	1.426 ± 0.801	7.944 ± 1.322
block	large	100	86.144 ± 10.469	76.539 ± 10.288	1.420 ± 0.737	8.185 ± 1.392
live	tiny	0	6.829 ± 00.237	5.076 ± 00.228	0.271 ± 0.049	1.482 ± 0.055
live	small	0	7.524 ± 00.204	5.779 ± 00.213	0.260 ± 0.059	1.485 ± 0.058
live	medium	0	8.228 ± 00.209	6.415 ± 00.219	0.343 ± 0.062	1.469 ± 0.029
live	large	0	9.417 ± 00.207	7.686 ± 00.198	0.341 ± 0.051	1.390 ± 0.086
live	small	10	12.790 ± 00.613	10.216 ± 00.589	0.187 ± 0.092	2.386 ± 0.140
live	tiny	10	11.864 ± 00.423	9.322 ± 00.325	0.263 ± 0.159	2.279 ± 0.454
live	medium	10	13.618 ± 00.628	11.043 ± 00.596	0.203 ± 0.030	2.371 ± 0.278
live	large	10	16.069 ± 01.568	13.710 ± 01.662	0.191 ± 0.035	2.168 ± 0.365
live	tiny	50	38.402 ± 02.483	32.889 ± 02.496	0.372 ± 0.093	5.141 ± 1.163
live	small	50	44.869 ± 03.322	39.080 ± 03.111	0.430 ± 0.058	5.359 ± 0.961
live	medium	50	49.298 ± 04.133	42.721 ± 04.234	0.519 ± 0.241	6.059 ± 1.577
live	large	50	56.299 ± 08.190	50.608 ± 08.585	0.417 ± 0.117	5.274 ± 1.321
live	tiny	100	69.905 ± 07.106	60.089 ± 07.093	0.809 ± 0.515	9.007 ± 1.120
live	small	100	79.181 ± 04.538	69.265 ± 05.341	0.665 ± 0.071	9.251 ± 1.798
live	medium	100	84.653 ± 06.314	75.539 ± 06.050	0.677 ± 0.087	8.437 ± 1.615
live	large	100	96.000 ± 12.398	86.239 ± 11.989	0.585 ± 0.201	9.176 ± 1.411

Table A.3: Migration durations sorted by drop rates with ascending flavor with confidence intevals with 95% confidence level

List of Figures

List of Tables

Bibliography

[1] "Amazon web services 'growing fast'," Apr. 2015. [Online]. Available: http://www.bbc.com/news/business-32442268

[2] J. T. Piao and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing," in *Grid and Cooperative Computing (GCC), 2010 9th International Conference on.* IEEE, 2010, pp. 87–92.

[3] E. Mohammadi, M. Karimi, and S. R. Heikalabad, "A novel virtual machine placement in cloud computing," *Australian Journal of Basic and Applied Sciences*, vol. 5, no. 10, pp. 1549–1555, 2011.

[4] C. Hyser, B. Mckee, R. Gardner, and B. J. Watson, "Autonomic virtual machine placement in the data center," 2008.

[5] P. Rygielski and A. Gonczarek, "Migration-aware optimization of virtualized computational resources allocation in complex systems," in *Systems Engineering (ICSEng), 2011 21st International Conference on.* IEEE, 2011, pp. 212–216.

[6] S. Kolb, J. Lenhard, and G. Wirtz, "Application Migration Effort in the Cloud–The Case of Cloud Platforms."

[7] N. M. Huber, "Autonomic Performance-Aware Resource Management in Dynamic IT Service Infrastructures," Ph.D. dissertation, Karlsruhe, Karlsruher Institut für Technologie (KIT), Diss., 2014, 2014.

[8] N. Huber, M. Von Quast, F. Brosig, and S. Kounev, "Analysis of the performance-influencing factors of virtualization platforms," in *On the Move to Meaningful Internet Systems, OTM 2010.* Springer, 2010, pp. 811–828.

[9] N. Huber, M. von Quast, M. Hauck, and S. Kounev, "Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments." in *CLOSER*, 2011, pp. 563–573.

[10] Q. Noorshams, A. Busch, A. Rentschler, D. Bruhn, S. Kounev, P. Tuma, and R. Reussner, "Automated Modeling of I/O Performance and Interference Effects in Virtualized Storage Systems," in *Distributed Computing Systems Workshops (ICDCSW), 2014 IEEE 34th International Conference on.* IEEE, 2014, pp. 88–93.

[11] Q. Noorshams, D. Bruhn, S. Kounev, and R. Reussner, "Predictive performance modeling of virtualized storage systems using optimized statistical regression

techniques," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. ACM, 2013, pp. 283–294.

[12] K. Diepold, "The Quest for a Definition of Quality of Experience," *Qualinet Newslet*, pp. 2–8, Oct. 2012. [Online]. Available: www.cost.eu/download/34536

[13] J. Rinne, M. Liljeberg, and J. Jouppi, "Quality of service definition for data streams," Jan. 15 2008, uS Patent 7,320,029. [Online]. Available: https://www.google.com/patents/US7320029

[14] T. Hoßfeld, R. Schatz, M. Varela, and C. Timmerer, "Challenges of QoE Management for Cloud Applications," *IEEE Communications Magazine*, vol. April issue, Apr. 2012.

[15] F. Wamser, "Performance Assessment of Resource Management Strategies for Cellular and Wireless Mesh Networks," Ph.D. dissertation, University of Würzburg, Jan. 2015. [Online]. Available: https://opus.bibliothek.uni-wuerzburg.de/files/11151/wamser_florian_resourceallocation.pdf

[16] T. Hoßfeld, R. Schatz, M. Seufert, M. Hirth, T. Zinner, and P. Tran-Gia, "Quantification of YouTube QoE via Crowdsourcing," in *IEEE International Workshop on Multimedia Quality of Experience - Modeling, Evaluation, and Directions (MQoE 2011)*, Dana Point, CA, USA, Dec. 2011.

[17] F. Wamser, T. Zinner, L. Iffländer, and P. Tran-Gia, "Demonstrating the Prospects of Dynamic Application-Aware Networking in a Home Environment," Chicago, IL, USA, Aug. 2014.

[18] F. Wamser, L. Iffländer, T. Zinner, and P. Tran-Gia, "Implementing Application-Aware Resource Allocation on a Home Gateway for the Example of YouTube," in *Mobile Networks and Management, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Würzburg, Germany, Sep. 2014.

[19] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[20] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI," *IEEE Internet computing*, no. 2, pp. 86–93, 2002.

[21] J. Turnbull, Dec. 2013.

[22] R. Vaupel, Q. Noorshams, S. Kounev, and R. Reussner, "Using Queuing Models for Large System Migration Scenarios–An Industrial Case Study with IBM System z," in *Computer Performance Engineering*. Springer, 2013, pp. 263–275.

[23] "OpenStack FAQ." [Online]. Available: https://www.openstack.org/projects/openstack-faq/

[24] E. Freeman, E. Freeman, B. Bates, and K. Sierra, *Head First Design Patterns*. O' Reilly & Associates, Inc., 2004.

[25] "TCPDump." [Online]. Available: http://www.tcpdump.org/

[26] "WireShark." [Online]. Available: https://www.wireshark.org/

[27] S. Oberste-Vorth, "Measurement-based Performance Comparison of SDN-related Northbound APIs," Master's thesis, University of Würzburg, Mar. 2015.

[28] "CirrOS." [Online]. Available: https://launchpad.net/cirros

[29] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018 (Proposed Standard), Internet Engineering Task Force, Oct. 1996. [Online]. Available: http://www.ietf.org/rfc/rfc2018.txt

[30] "GNU Wget." [Online]. Available: https://www.gnu.org/software/wget/

[31] "nginx." [Online]. Available: http://nginx.org/

[32] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[33] "ffmpeg." [Online]. Available: https://www.ffmpeg.org/

[34] "Sintel." [Online]. Available: https://durian.blender.org/download/

[35] "MPlayer." [Online]. Available: http://www.mplayerhq.hu/

[36] L. De Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo, "TAPAS: a Tool for rApid Prototyping of Adaptive Streaming algorithms," in *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*. ACM, 2014, pp. 1–6.

[37] "Counter-Strike: Source." [Online]. Available: http://store.steampowered.com/app/240/

[38] M. Hitchens, "A Survey of First-person Shooters and their Avatars," *The International Journal of Computer Game Research*, vol. 11, no. 3, December 2011. [Online]. Available: http://gamestudies.org/1103/articles/michael_hitchens

[39] R. Voith and S. Sudhaman, "Method and apparatus for interleaving data in an asymmetric digital subscriber line (ADSL) transmitter," Apr. 7 1998, uS Patent 5,737,337. [Online]. Available: https://www.google.com/patents/US5737337