# Proceedings of the Third HPI Cloud Symposium "Operating the Cloud" 2015

David Bartok , Estee van der Walt, Jan Lindemann,
Johannes Eschrig, Max Plauth (Eds.)

**Technische Berichte Nr. 105**

des Hasso-Plattner-Instituts für
Softwaresystemtechnik
an der Universität Potsdam

Universität Potsdam

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 105

David Bartok | Estee van der Walt | Jan Lindemann | Johannes Eschrig |
Max Plauth (Eds.)

# Proceedings of the Third HPI Cloud Symposium “Operating the Cloud” 2015

Universitätsverlag Potsdam

**Bibliografische Information der Deutschen Nationalbibliothek**
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de/ abrufbar.

**Universitätsverlag Potsdam 2016**
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URN urn:nbn:de:kobv:517-opus4-87548
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-87548

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-360-2

# Preface

Every year, the Hasso Plattner Institute (HPI) invites guests from industry and academia to a collaborative scientific workshop on the topic ″Operating the Cloud″. Our goal is to provide a forum for the exchange of knowledge and experience between industry and academia. Hence, HPI's Future SOC Lab is the adequate environment to host this event which is also supported by BITKOM.

On the occasion of this workshop we called for submissions of research papers and practitioner's reports. ″Operating the Cloud″ aims to be a platform for productive discussions of innovative ideas, visions, and upcoming technologies in the field of cloud operation and administration.

In this workshop proceedings the results of the third HPI cloud symposium ″Operating the Cloud″ 2015 are published. We thank the authors for exciting presentations and insights into their current work and research. Moreover, we look forward to more interesting submissions for the upcoming symposium in 2016.

# Contents

# Dependable Cloud Computing with OpenStack

Johannes Eschrig, Sven Knebel, Nicco Kunzmann

Operating Systems and Middleware Group
Hasso Plattner Institute Potsdam
first.last@student.hpi.de

Offering infrastructures as a service by means of cloud computing is gaining popularity. High availability aspects of these cloud computing systems are of great importance, as outages can be extremely costly. Setting up a cloud computing environment is very complex, thus making dependability testing non trivial. In our work, we introduce a system for installing a virtual OpenStack cloud computing environment and running dependability experiments on it. The installation as well as the experiments are automated in order to achieve reproducible test results as easily as possible. We propose a first selection of experiments for our testing framework and describe the results.

## 1 Introduction

As cloud computing becomes more and more popular, there is an increasing number of implementations to offer various cloud-service models like infrastructure as a service (IaaS), platform as a service (PaaS) or software as a service (SaaS). While many companies offer commercial solutions like the Amazon Elastic Compute Cloud (EC2) or HP Helion, there are also open source alternatives that can be freely installed and configured to meet the needs of ones projects with respect to the underlying hardware available.

One of the open source variants for achieving a cloud computing system is OpenStack. OpenStack is a cloud software stack which allows for offering infrastructure as a service, almost independent of the underlying hardware setup. OpenStack itself can be seen as a collection of services that can be setup depending on the specifications of the planned use cases. The most important components that OpenStack offers are the networking, virtualization and storage services. Furthermore, it is possible to add further components to an OpenStack installation, e.g., services that handle billing or allow for object storage in the cloud.

This paper will describe the results of the masters project "Dependable Cloud Computing with OpenStack" of the summer term 2015 at the Hasso Plattner Institute Potsdam. An important factor, especially in cloud computing, is dependability. When offering such a service, it should be highly available, meaning that the system should be continuously operational without failing. Therefore, our main task was to analyse dependability mechanisms of OpenStack. To do this, we chose to manually setup a clean OpenStack environment (i.e. none provided by a third party like HP Helion) on which we would be able to run the specific analyses. We turned this manual installation into an automated one in order to simplify and speed up the process of setting up a working OpenStack test environment and making the resulting analyses of dependability reproducible. Since no OpenStack installation is exactly the same, the reproducibility of the results of such analyses is not an easy feat. We tackle this issue by making the test environment for the experiments completely virtual. Thus we circumvent tedious hardware setup, hardware errors that disturb the experiments. This also allows a fast rerun of the experiments and switching off network infrastructure.

# 2 Related Work

In this chapter we will introduce work related to our masters project. In a first part we will describe work related to OpenStack and its possibilities of installation. The second part will cover the related work to dependability in OpenStack.

## 2.1 OpenStack Installation

In order to analyse the dependability of OpenStack, it is necessary to install an OpenStack instance. Due to the fact that such an analysis might require a clean and fresh OpenStack installation after each test run, a quick installation is of advantage. Further, independence of underlying hardware is vital to reproduce results. Merging the requirements of an easy and quick installation that achieve reproducible results leads us to look for possibilities to automatically install OpenStack completely in a virtual environment. There are various OpenStack derivatives both commercially and freely available.

*HP Helion*[1] is available both as a commercial-grade edition and a free-to-license community edition. The latter is available on promotional USB drives given out by HP. The HP Helion community edition installation is made to provide an easy installation routine with little need for configuration by means of such an USB drive. Further, it is possible to install HP Helion as an all-in-one system on virtual machines in addition to deploying it on bare-metal.

*DevStack*[2] is another possibility for an easy OpenStack installation. It is a development environment for OpenStack. Being designed for development on OpenStack, it is mainly used for an one-node installation of OpenStack. Additionally, DevStack also offers an option for a multi-node setup.

A manual installation of an OpenStack instance can take very long and be quite cumbersome, depending on the setup one is aiming to achieve. It is therefore of great advantage to automate the installation. As an OpenStack installation is distributed among a number of nodes, using an orchestration tool like Ansible[3] is advisable. Ansible manages nodes using SSH and Python.

[1]`http://www8.hp.com/us/en/cloud/hphelion-openstack.html.`
[2]`http://docs.openstack.org/developer/devstack/.`
[3]`http://www.ansible.com/.`

*Openstack-Ansible*[4] is an existing automated OpenStack installation project, which installs OpenStack on Vagrant virtual machines. We ran the installation script of this project, however encountered some bugs. In order to understand the underlying mechanisms of OpenStack ourselves, we decided to follow a similar approach to openstack-ansible based on KVM virtual machines.

## 2.2 Evaluating OpenStack Dependability

Due to the complexity and variety of possibilities to set up an OpenStack system, evaluating the dependability of OpenStack in general is no easy task. For this reason, we have decided to make simplifying assumptions about an OpenStack installation and define a test environment on which we can then run dependability experiments. An more general approach is also possible, i.e. building a framework for injecting faults into various OpenStack deployments, as was done for example by [3] or [2]. Both works thereby created a frameworks for injecting faults into OpenStack. [3] follows a similar approach to that of our masters project and uses a virtual environment for the setup of OpenStack, however only implements one simulated failure as proof of concept. [2] on the other hand focuses more on the fault injection aspect, especially targeting service communications, uncovering 23 bugs in two OpenStack versions. In our masters project, we aim to provide a framework for the evaluation of the cloud system OpenStack with the advantage of a fast and easy virtual installation of the system itself and easily extendable experiments for dependability testing.

The previous masters project on OpenStack also gave some insights on fault tolerance of OpenStack in [1], presenting a fault tree based on the high availability setup presented by [4].

# 3 OpenStack Test Environment

As described in Chapter 2, we tried various possibilities to install an OpenStack system on a virtual environment. In this chapter we outline the challenges we faced with these possibilities, ultimately leading to the decision to create our own installation routine for a virtual OpenStack environment. Also, we will describe this test environment in detail.

## 3.1 Existing OpenStack Installation Possibilities

We first tried installing the cloud computing environment HP Helion community edition. It promises an easy installation routine with little need for configuration. Further, it is possible to install HP Helion as an all-in-one system on virtual machines in addition to deploying it on bare-metal. This is an advantage with respect

[4]`https://github.com/openstack-ansible/openstack-ansible.`

to our requirement of achieving a virtual test environment for reproducible test results. However, even though HP Helion has its advantages for setting up one's own IaaS system for a real use scenario, we came to the conclusion that it is not suitable for our needs. The installation of HP Helion took around 90 minutes on our hardware, which is not feasible for repeated installations. Further, we found that HP Helion does not survive a reboot of the host or the virtual machines it is running on. Fixing this issue would have required understanding the underlying installation scripts, which would still not have been beneficial in understanding OpenStack itself. Additionally, with our limited knowledge of HP Helion, it would have been a challenge to customize the system to fit our needs. We thus concluded that we would not use HP Helion for analyzing the dependability of OpenStack in line with this masters project.

A further option for an OpenStack deployment to work with for dependability testing we considered was DevStack. Due to the nature of the use cases for which DevStack is made, multi-node and high-availability setups are not the main focus, and therefore not documented well enough for us to customize DevStack and use it for dependability analyses. Also, it is not possible to generalize results of dependability analyses run on DevStack to a full OpenStack installation, as DevStack is not designed with real deployment in mind. As a result, we decided to use a full OpenStack installation for running our analyses.

## 3.2 Specifying our own OpenStack Environment

In order to be able to make the OpenStack test environment installation as easy and quick as possible, so that one can concentrate on the dependability analysis, we chose to install OpenStack virtually. A further advantage of this virtual installation is the reproducibility of experiment results, which is important to be able to make scientific statements about the dependability of OpenStack. We used libvirt[5] to create a number of virtual machines based on simple configuration files and Ansible[6] to orchestrate the installation of OpenStack on these nodes. The details of this automated installation are described in Chapter 4.

In order to create an useful test environment for dependability experiments, it was necessary to define an architecture. We chose this architecture to be a simplified OpenStack instance, meaning that we focus only the most important of the OpenStack services available. This can be seen as a bottom-up approach, as we focus on evaluating a simpler system than one might encounter when looking at an OpenStack system in production mode. One advantage of this approach is that, due to the simplicity, it is easier to make statements about OpenStack in general, than on one specific system. Libvirt and Ansible allow to add more nodes, create a more complex OpenStack system and, extend the proposed architecture by means of high availability mechanisms. We then draw conclusions about their effectiveness.

[5]`http://libvirt.org/`.
[6]`http://www.ansible.com`.

Figure 1 shows our virtual test environment architecture, which is the proposed architecture of the official OpenStack install guide. [7] All nodes are virtual machines running on a physical host. The tenant virtual machines are dispatched on the compute node by means of nested virtualization. By default, our environment contains the following nodes:

- One controller node: this node runs the OpenStack Dashboard (Horizon), the API services, the MySQL database, the RabbitMQ message queue server, the scheduler for the compute resources, Identity (Keystone) and Image (Glance) services.
- One network (Neutron) node: this node handles the internal and external routing and DHCP services for the virtual networks.
- Two compute nodes: the compute nodes are the computing resources for running the virtual machines of the OpenStack users. They run the hypervisor and services like `nova-compute`, which is responsible for creating and terminating virtual machine instances through the hypervisor APIs.
- Two object storage (Swift) nodes: these nodes operate the OpenStack container and object services and each contain two local block storage disks for persisting the objects.

Further, the following networks are used to communicate between nodes and instances:

- Management network: this network is used for the OpenStack administration, i.e., it connects the OpenStack services on the different nodes.
- Tenant or tunnel network: these networks can be created by the OpenStack users to achieve communication between projects or instances.
- External network: this network provides internet access to the instances.

This architecture is comprehensive enough to test various OpenStack use cases and analyze the dependability of the system.

The virtual OpenStack installation requires far less hardware resources than a distributed bare metal installation. It is possible to install a fully functional simplified test environment with one compute and object storage node on a quad-core Intel Xeon machine with 8GB RAM. For the full installation, more resources are recommended. We used a 16-core machine with 64GB RAM.

[7] `http://docs.openstack.org/kilo/install-guide/install/apt/content/index.html.`

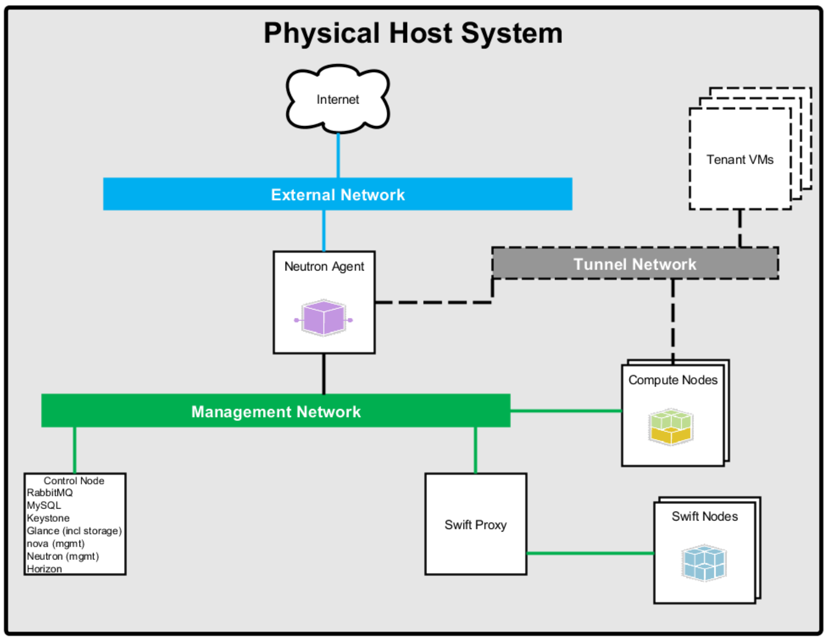

**Figure 1:** Our test environment architecture, based on the one proposed by the OpenStack install guide

## 4 Automated Installation of OpenStack

In this chapter, we describe the automated installation process of OpenStack on our virtual environment. We give an introduction to the usage and a conceptual overview.

### 4.1 How to Install OpenStack using our System

The installation scripts are developed and tested on a Ubunutu 14.04 LTE (Trusty Tahr) desktop version. All required dependencies (e.g., Ansible, libvirt, etc.) are installed automatically, thus an internet connection is required. It installs OpenStack virtually creating the architecture described in Chapter 3.2. The virtual machines are automatically created. The installation takes between 10–15 minutes.

After the successful installation, a snapshot named "initial" is created. This allows for thorough dependability testing without re-installing the whole system after each experiment. Snapshots can be created manually as well. The snapshotting mechanism will shut down all virtual machines, snapshot the virtual hard drives of all nodes, and bring back up all machines.

## 4.2 Creating the Virtual Environment

The virtual environment, i.e., the virtual networks and the virtual machines, are defined in the `config` folder as libvirt network and libvirt domain XML-files. These XML-files define for example the IP addresses of the networks or the hardware specifications (size of RAM, number of cores, virtual hard drives, network interfaces) of the virtual machines. The virtual machines are then created with an Ubuntu cloud image[8], which are pre-configured images customized especially for running on cloud platforms.

The initialization of the virtual machines is done using cloud-init[9], which allows for setting passwords and SSH keys to aid seamless connection thereafter, which is also a prerequisite for utilizing Ansible in the next steps. Further, using cloud-init, the correct `etc/network/interfaces` configuration files are copied to the virtual machines.

In addition to the virtual machines for the OpenStack nodes, a further virtual machine named "aptcache" is created. This machine is used as a package repository by the others. The packages it delivers are not updated, meaning that the versions of all installed packages are frozen. This is important for acquiring a reproducible test environment and prevents different experiment outcomes to be caused by different package versions throughout the system.

These virtual machines create the base of the virtual OpenStack test environment and are now ready for the actual OpenStack installation.

## 4.3 Installing OpenStack on Virtual Machines with Ansible

In order to be able to install OpenStack on the created virtual environment, Ansible must be configured in such a way that virtual machines are grouped. This allows for installing different parts of OpenStack on the different nodes. These groups are defined in a so called Ansible hosts file, which assigns the IP addresses of the different nodes to different groups. In our case, each node type is a group, i.e., "controller", "network", "compute" and "object". This allows for the so called Ansible playbooks to be executed on the specified groups of nodes in parallel.

The Ansible playbooks that run the installation of OpenStack on the virtual nodes are strongly based on the official OpenStack installation guide[10], which is very extensive. This gives the advantage of not having to write any further OpenStack related documentation additionally to the documentation of the technicalities of the creation of the virtual machines and the Ansible installation. The arrangement of the Ansible playbooks in a folder structure derived from the content of the OpenStack documentation allows for easily finding the corresponding part of

[8] `https://cloud-images.ubuntu.com/trusty/`.

[9] `https://cloudinit.readthedocs.org/`.

[10] `http://docs.openstack.org/kilo/install-guide/install/apt/content/index.html`.

the documentation should one require information about a certain part of the installation.

The Ansible installation of OpenStack starts with an initial preconfiguration of the virtual nodes. In this step, the hosts files are created in order to be able to connect to the nodes by their host names. These host names are then also added to the SSH `known_hosts` files to enable the SSH connection without warning messages. Further steps include setting the locale of the virtual machines to prevent locale errors as well as deactivating the `/etc/cloud/cloud.cfg` file, as all configuration of the images is done in the previous step, see Chapter 4.2.

A special virtual machine named "aptcache" is set up first and independently of the others. It runs Apt-Cacher-NG[11], a caching proxy for Linux package repositories. All other VMs are set up to request their packages through it. This a) decreases network traffic and wait times and b) can be used to repeat the setup process while using the exact same package versions as the first time: The first install seeds the package cache, a repeated installation then can receive all packages from the cache instead of fetching potentially newer versions from upstream. To allow this, the cached data is not stored in the VM, but in a folder shared from the host machine.

The first step of the actual OpenStack installation is installing the basic environment. This includes adding the OpenStack package repository to the nodes and installing the MySQL database on the controller and the message queue RabbitMQ on all nodes.

The next step is the installation of the OpenStack identity service (Keystone) on the controller node. This service is responsible for the permissions of users and keeping track of the available OpenStack services with their endpoints. The installation of this service includes creating a database, installing the Keystone client packages and populating the database. A demo and an admin tenant are initially created. As with all OpenStack services, the installation is finalized by creating the service entity and the API endpoint.

Next, the image service Glance is added on the controller node. This service allows for retrieving and registering virtual machine images. Again, a database for this service is created along with the installation of the Glance client packages and the creation of an API endpoint.

OpenStack compute is then setup on the controller and the compute nodes. This component is responsible for the administration and hosting of the computing systems. It allows among other things for creating and terminating virtual machine instances through hypervisor APIs and is responsible for scheduling on which compute node an instance should run. To install the compute service Nova on the controller, the respective database and API endpoint is created and the nova service is configured. On the compute nodes, the nova-compute packages are installed and configured to run a QEMU hypervisor with the KVM extension.

The OpenStack networking Neutron components are installed next on the network, controller and compute nodes. The main responsibility of the networking

[11]`https://www.unix-ag.uni-kl.de/~bloch/acng/`.

component is to provide connectivity between the instances running on the compute node. On the controller node, database and endpoint API are created, the networking server component is configured and the Modular Layer 2 plug-in is configured. This plug-in is responsible for the networking framework of the instances running on the compute nodes. On the network node, the networking components are installed and configured accordingly. The layer-3 agent for routing services, the DHCP agent, the metadata agent and the Open vSwitch service are configured. Lastly, the networking components, the Modular Layer 2 plug-in and the Open vSwitch service are configured on the compute node. The external and tenant networks are then created to finalize the installation.

The OpenStack web interface dashboard Horizon is then installed on the controller node. This allows administrators and users to access and manage their resources on the OpenStack cloud.

The last component that is added to OpenStack in regards to this masters project is the object storage Swift. This allows to create containers, upload and download files and management of the objects on the storage nodes. On the controller node, the proxy service that handles the requests for the storage nodes is installed and configured. The storage nodes require two empty storage devices for persisting the objects. The virtual machines for these nodes contain two virtual devices in the qcow file format. The OpenStack Swift components are then installed and configured on the object storage nodes. Following the installation, the initial account, container and object rings are created.

# 5 Running Dependability Experiments

In this chapter, we will describe our OpenStack dependability experiments and their results. The implementations of these experiments all follow the same structure, so that it is easy to add further experiments if needed. In our experiments, we focused on covering the failure of various OpenStack components or nodes. We will give insights on how these failures affect the OpenStack environment and how the system deals with each fault. This serves partly to show weak points of the setup and partly to document details about its behaviour.

Experiments consist of multiple stages: The *setup* stage creates all elements necessary to run the experiment. The *break* stage then breaks things. An optional *heal* stage tries something simple to undo the damage (e.g. reboot a shutdown node). After each of this stages, a *check* step is executed, which observes the state of the system and reports its findings to the user. Generally, after the setup stage all checks should be successful. Where user observations are useful (e.g. by looking not just at API results, but seeing how Horizon represents situations), the user is prompted to do so.

Using the snapshotting mechanism, the user can always completely restore the system, even if it didn't survive an experiment. This is not done by default because a) it takes some time and is not always necessary and b) to allow the user to inspect the system state after the experiment has concluded, e.g. to find or work out steps to fix remaining issues.

## 5.1 Experiment Results

This section describes our four example experiments and discusses their results.

**Experiment 1: Control Node crash**
In this experiment, a crash of the control node in the system is simulated.

**Technical Background** The controller node stores global information of the OpenStack cluster and runs the sub-services depending on it, which then give services on other nodes the specific information they need to e.g. run a specific instance or to build network connectivity. In our setup, it also provides the dashboard Horizon and runs the authentication service. A failure of this node obviously is going to have a large impact, but some things that already are set up on other nodes continue to operate.

**Experiment** The experiment script creates an instance and then uses ping to verify availability of both the compute node and the started instance. It also tries to access OpenStack APIs. To simulate the fault, it either issues a shutdown command, simulating the unavailability of the node to the controller or more severely, just turns off its infrastructure virtual machine to simulate a full crash.

**Results and Conclusion** While the control node is turned off, outside connectivity to the already created instance remains, since it runs on the compute node and its network connection to the outside (managed by Neutron, via the network node) is unaffected. On the other hand, all attempts to use OpenStack APIs fail. Most user-facing APIs are accessed via the controller and thus completely unavailable. Others report errors, since every action has to be authorized using the Keystone service, which only runs on the compute node in our setup.

After the controller node is up again, OpenStack takes a few minutes to re-establish all service connections and in most cases is fully operational again. It is possible that the compute instances fail to reconnect to RabbitMQ and their services have to be restarted manually.[12]

This shows that already running instances on OpenStack are generally not impacted by temporary failure or maintenance of quite a few central OpenStack components, but during their unavailability no changes can be made and recovery might need manual intervention by an operator.

If the controller node was shut down hard, obviously many more failure scenarios related to the underlying operating system or services are possible e.g. missing information in databases or damaged file systems.

[12] http://docs.openstack.org/openstack-ops/content/maintenance.html#cloud_controller_storage.

**Experiment 2: Memchached Service Loss of Data**
This experiment shows the effects of Keystone losing authentication tokens due to the design of its storage mechanism.

**Technical Background** A common way to authenticate for operations against the OpenStack API is to use tokens. A more complex and privileged authentication (e.g. a password check) is done once to obtain a security token. These tokens have a limited lifetime and can be limited in scope, so a user can generate a token for a specific task and pass it to a service, which then can use it to access other services in the users name. Tokens also are internal to OpenStack, whereas other authentication might require accessing an external authentication provider (e.g. an LDAP server).

The Keystone service stores these tokens in memcached[13], which is, as the name alludes to, an in-memory caching service. Designed as just a fast caching layer, it neither has persistence to disk nor does it guarantee to keep all data it stores. If it deems necessary it can evict any information at any time (i.e. because it is under memory pressure).[14]

**Experiment** The admin credentials are used to create a token. To verify the tokens validity, it is used to authenticate an operation against the Keystone API. To cause memcached to evict it from the cache, the command for memcached to delete all its data is issued.

**Results and Conclusion** Subsequent attempts to use the token fail, since it has been deleted. This shows the consequences of OpenStack using an unreliable data store to save a central element of its authentication system, instead of using memcached as designed only as a cache to improve lookup speeds.

If a user is logged into the Horizon dashbord while the experiment is running, it also sometimes allows to observe failures: The dashboard site is still accessible (because the session on the web server still exists), but no information from OpenStack is shown because attempts to retrieve it fail due to the invalid token. The user has to log out and back in again to create a new token.

**Experiment 3: Compute Nodes Unavailable**
This experiment documents the behaviour if connectivity to compute nodes is lost.

**Technical Background** Instances are distributed among the compute nodes by the Nova scheduler, which runs on the controller node. Inspecting the state of VMs e.g. directly via the Nova API or via the Horizon Dashboard is also done via information stored by Nova on the controller node, where it collects information from all compute hosts. If the controller looses connectivity to compute nodes, it can't accurately report the state of individual instances, as seen in this experiment.

**Experiment** This experiment first creates an instance to observe throughout the different stages. Then the first level of fault is introduced: All compute nodes are

[13] `http://memcached.org/`.
[14] `https://code.google.com/p/memcached/wiki/NewUserInternals#How_the_LRU_Decides_What_to_Evict`.

removed from the management network. Then an attempt to start a second instance is made. After this, a more severe fault is created by shutting down all compute nodes. Then a *fix* is attempted by restarting the VMs.

**Results and Conclusion** After the first fault, Nova on the controller has no connectivity to the compute nodes and is therefore unable to actually start the second instance. The first instance is still reported as being active, which in this case is correct: it is still running and can be reached from the outside network. Since it already has lost all connectivity to the compute nodes, things look exactly the same after the compute nodes are actually shut down, but in this case the state information is wrong: the VMs obviously are not active, but shut down together with the host they were running on. After the reboot of the compute nodes, they reconnect to the controller and the state of all VMs is reported correctly again.

This shows that when Nova looses connectivity to a compute node it can't report accurate information, which is one of the reasons why Nova doesn't implement functionality to automatically restart such VMs. Short interruption of Nova services, e.g. due to software updates, do not necessarily disrupt instance operations. Such decisions are left to other systems that can collect more information (e.g. by running active tests against VMs) and can be configured for specific application needs, like OpenStack Heat.

We find it interesting that the developers choose to report such instances as active and did not introduce another state to represent this special situation.

## 6 Conclusion

In our masters project we created a platform for the automated installation of an virtual OpenStack environment as well as a framework for some first dependability experiments on this environment.[15] The advantage of our platform is the very fast installation (10–15 minutes) of a complete OpenStack environment on very limited hardware compared to a full bare metal installation. This makes running dependability experiments comparatively easy. Features like snapshotting would not be available on a bare metal setup, but are very useful when repeating such experiments. Further, our platform is easily extendable, both for adding further OpenStack components as well as further experiments.

These features can be the foundation for future work. Due to the limited time and personal resources, we were not able to implement the installation of a full OpenStack high availability setup. Such a setup could make it possible to compare the dependability of the "normal" OpenStack setup to the high availability one, by running the experiments on both. Further, due to the fact that we use Ansible for the installation, the playbooks themselves can be easily used for installing OpenStack on bare metal. For this, the scripts for configuring the virtual environment could be

[15]`https://github.com/MasterprojectOpenStack2015/sourcecode.`

extended in order to make it possible to configure a bare metal setup. This would allow running the experiments on a bare metal OpenStack installation.

## References

[1] M. Bastian, S. Brueckner, K. Fabian, M. Hopstock, D. Korsch, and D. Stelter-Gliese. *Cloud Computing with OpenStack*. Report Masters Project. 2015.

[2] X. Ju, L. Soares, K. G. Shin, K. D. Ryu, and D. Da Silva. "On Fault Resilience of OpenStack". In: *Proceedings of the 4th Annual Symposium on Cloud Computing*. SOCC '13. Available at `http://doi.acm.org/10.1145/2523616.2523622`. Santa Clara, California: ACM, 2013, 2:1–2:16. DOI: `10.1145/2523616.2523622`.

[3] M. Kollárová. "Fault injection testing of OpenStack". Available at `http://is.muni.cz/th/325503/fi_m/`. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno, 2014.

[4] Q. Teng. *Enhancing High Availability in Context of OpenStack*. Available at `https://www.openstack.org/summit/openstack-summit-atlanta-2014/session-videos/presentation/enhancing-high-availability-in-context-of-openstack`. 2014.

# Protecting Minors on Social Media Platforms - A Big Data Science Experiment

Estée van der Walt, J.H.P. Eloff

Department of Computer Science
University of Pretoria, South Africa
estee.vanderwalt@gmail.com;eloff@cs.up.ac.za

Interpersonal communications on social media, hosted via cloud computing infrastructures, has become one of the most common online activities. This is especially so for children and adolescents (minors) who may be accidentally and intentionally exposed to cyber threats such as cyber bullying, pornography and pedophilia. Most of these unwanted activities deal with some form of identity deception. This paper presents work-in-progress that leverages on the advances made in big data and data science to assist in the early detection of identity deception and thereby to protect minors using social media platforms.

## 1 Introduction

Facebook, Twitter, MySpace and SnapChat host the online activities of millions of individuals spanning across various age groups. According to a study in the USA [12] done on the online activities of minors, children and adolescents, it is showed that 85 percent of 18–29 year olds use social media platforms. These numbers however mostly exclude minors as many social media sites have enacted age based bans to comply with laws like COPPA [19]. Minors may be accidentally and intentionally exposed to cyber threats such as cyber bullying [19] and pedophilia [7]. Many of these threats imply some form of identity deception [2]. Of particular interest to this study is the case of counterfeiting an identity. It is easy for a predator to counterfeit an identity and to go unnoticed in a big data environment, such as social media platforms [5]. There is a need for new innovative solutions that can minimize the risk of identity deception on social media platforms.

The remainder of this paper is structured as follows: Section 2 describes background and related research whilst Section 3 presents work-in-progress, in the form of an experiment. The experiment shows how big data and data science can be leveraged to construct an Identity Deception Indicator. Lastly Section 4 will conclude the discussion at hand.

## 2 Background

### 2.1 Big Data and Data science

Big data, like the micro-blogs from Twitter, is hosted on cloud computing infrastructures due to its size, need for availability and complexity [11].

- The three main characteristics of big data is aptly defined as the 3V's [13].
- Volume; Minors are actively contributing content on social media daily [19].
- Velocity; Many minors actively participate in online gaming. Within these games speed and accuracy is critical for stepping out as the victor [7].
- Variety; Most minors will not only write text on social media but also contribute in the form of videos or photos.

Many other characteristics of big data have been proposed like 'Value' [6], 'Viability' [3], 'Validity' [1] and 'Veracity' [9]. According to Provost and Fawcett 2013, Data Science "involves principles, processes, and techniques for understanding phenomena via the automated analysis of data" [21]. It appears that the protection of minors against people with harmful intentions in online communities are of evolving nature [16] and that the availability of sufficient test, sample and training data sets are limited [18].

### 2.2 Cyber-security

According to ISO/IEC 27032 Cyber-security is the safeguarding of an individuals or a society's interest whilst interacting in cyber space [24]. From a vulnerability point of view, humans in general and minors, in particular, are not good in detecting counterfeit identities. Minors are usually easy targets. Social media platforms provide the ideal platform for an attack [26] mainly because of its big data nature and the complexity of non-textual data. Most existing countermeasures are based on plug-ins for safe-browsing on the internet [18]. These countermeasures are however inadequate for detecting identity deception.

### 2.3 Human factors

Yet Hargittai, Shultz and Palfrey [10] report that many minors under the age of 13 use social media sites like Facebook and lie about their age during site registration. It even reaches as far as parents registering on behalf of their children on social media sites [15]. Parents are however still concerned with their children meeting strangers online [7] and as at this moment there is no control or safe guard against this [12].

The problem of protecting minors are even more complicated by the fact that many minors lie about their age whilst communicating online [14, 20, 19]. It is easy to impersonate someone else on a Facebook account for example to slander their image [7] or lie in a chat or micro blogging site like Twitter about your age. It can therefore be stated that from a human factor point of view the challenge is to determine the authenticity of a person's identity.

**Authenticity of an identity**
A number of measures for determining the authenticity of an identity are based on a so-called 'identity score' [25]. Identity scores use information like personal data, public records, Internet data, government records and predictive behavior patterns to determine the authenticity of a person's identity.

Other research efforts to identify the age of an online identity includes the use of sentiment or emotions expressed in micro blogs and ngram counts, to understand whether a person is a pedophile or not, was performed in a study by Bogdanova [5], detecting age on Facebook through the use of vocabulary [22] and searching for pattern matching rules where a number followed by 'yr' or 'year' could denote a person describing their age [23]. Except for a US patent [4], none of the existing research efforts weighted the importance of certain identity attributes over another in the process of detecting identity deception.

# 3 Big Data Science Experiment

## 3.1 Process

The big data science experiment proposed to identify identity deceptions. Figure 1 shows the process followed for conducting the full experiment.

The remaining discussion that follows only focus on the first 6 steps of the process as outlined in Fig. 1.

## 3.2 Determine experiment objective

The objective of the experiment is to explore with creating an identity deception indicator with which minors could be protected on social media platforms.

## 3.3 Identify the source of social media data

Twitter was chosen as the source for big data. Twitter has over 1 billion subscribers for 400 million actively tweeting every day [8, 17]. Twitter is a dependable and realistic medium for research as mentioned by Durahim [8]. The main reason is the ease with which data can be freely retrieved from the cloud. Twitter has made a REST and streaming API available to developers for this purpose. The free service does however limit the amount of requests to the API to 180 per 15-minute window.

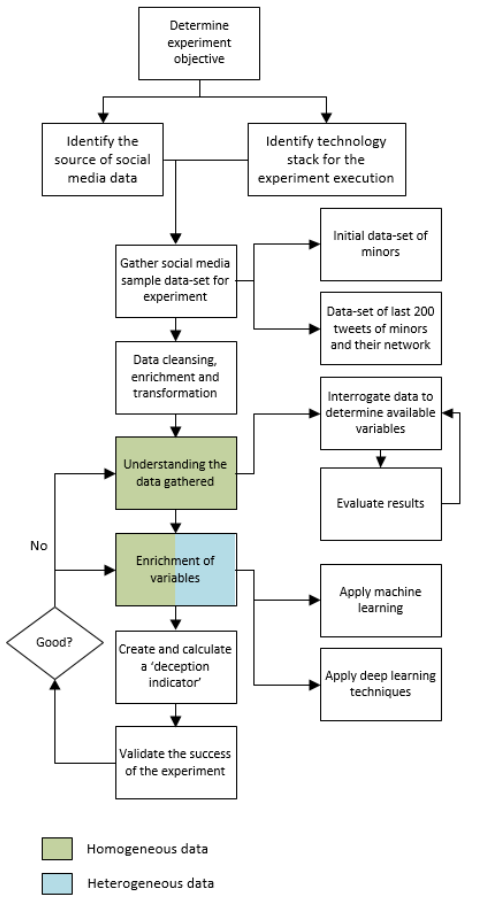
Determine experiment objective
Identify the source of social media data
Identify technology stack for the experiment execution
Gather social media sample data-set for experiment
Initial data-set of minors
Data-set of last 200 tweets of minors and their network
Data cleansing, enrichment and transformation
Interrogate data to determine available variables
Understanding the data gathered
Evaluate results
No
Enrichment of variables
Apply machine learning
Good?
Create and calculate a 'deception indicator'
Apply deep learning techniques
Validate the success of the experiment
Homogeneous data
Heterogeneous data

**Figure 1:** Big Data Science experiment

For the purposes of the experiment, the authors have decided to retrieve tweets from Twitter users who have marked their tweets with a hash tag indicator of 'school' or 'homework'. Schwartz et al [22] determined that these two words are most common with minors between the ages of 13 and 18. Although Twitter has no indication of age, it is expected that by using these hash tag indicators it should be possible to get an initial sample set, referred to as Set-1, of minors as a big data-set. Following the retrieval of Set-1 an additional data-set, referred to as Set-2, was retrieved containing, amongst other information, the last 200 tweets for each twitter user in Set-1 as well as the followers for that user. The final data-set, a combination of Set-1 and Set-2, contains the initial tweets with hash tag indicators 'school' or 'homework' as well as a history of previous tweets.

## 3.4 Identify technology stack for the experiment execution

Figure 2 illustrates a high level overview of the technology stack used for conducting the experiment. However, for the results as discussed in this paper not all components of the stack were used.

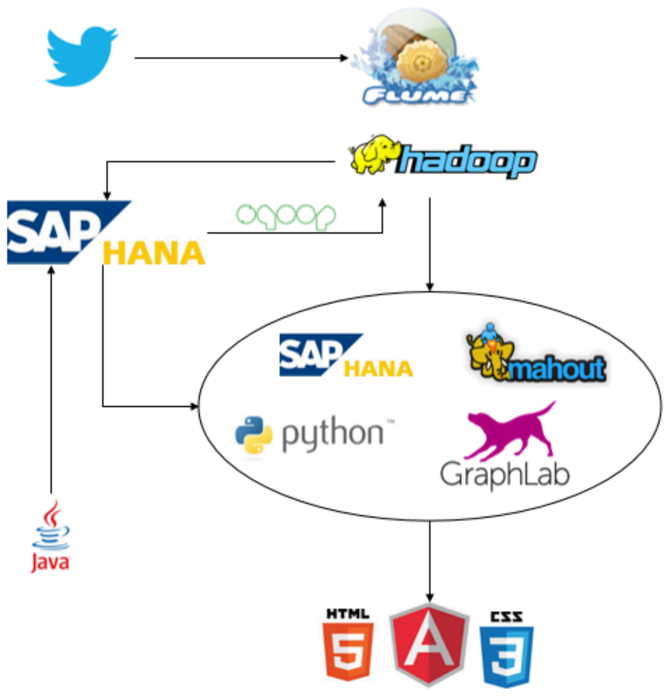

**Figure 2:** Technology stack for experiment

Some of the main components are discussed next:

- Twitter: The Twitter4j Java API was used to dump the data needed for the experiment in a big data repository.
- Hadoop: For the purposes of this experiment HDP Hadoop runs on an Ubuntu Linux virtual machine hosted in "The HPI Future SOC" research lab in Potsdam, Germany. This machine contains 4TBs of storage, 8GB RAM, 4 x Intel Xeon CPU E5-2620 @2GHz and 2 cores per CPU. Hadoop is well known for handling heterogeneous data in a low-cost distributed environment, which is a requirement for the experiment at hand.
- Flume: Flume is used as one of the services offered in Hadoop to stream initial Twitter data into Hadoop and also into SAP HANA.
- Sqoop: This service in Hadoop is used to pull data from SAP HANA back to the Hadoop HDFS. Analytical results e.g. machine learning and predictive modeling for the experiment will be generated on both Hadoop and SAP HANA. These results will be stored on both platforms and Sqoop facilitates this requirement.
- SAP HANA: A SAP HANA instance is used which is hosted in "The HPI Future SOC" research lab in Potsdam, Germany on a SUSE Linux operating system. The machine contains 4TBs of storage, 1TB of RAM and 32CPUs /

100 cores. The in-memory high-performance processing capabilities of SAP HANA enables almost instantaneous results for analytics.

- The XS Engine from SAP HANA is used to accept streamed Tweets and populate the appropriate database tables.

## 3.5 Gather social media sample data set for experiment

To be able to define a store as 'big data' anything from 2 to 4TB of test data from Twitter will be retrieved for the experiment. Keeping the Twitter rate limit, 180 requests per 15-minute window, in mind it was found that initially 4,344 tweets on average per hour could be retrieved. With code optimization this rate was later improved to 46,891 tweets on average per hour as seen from Fig. 3.

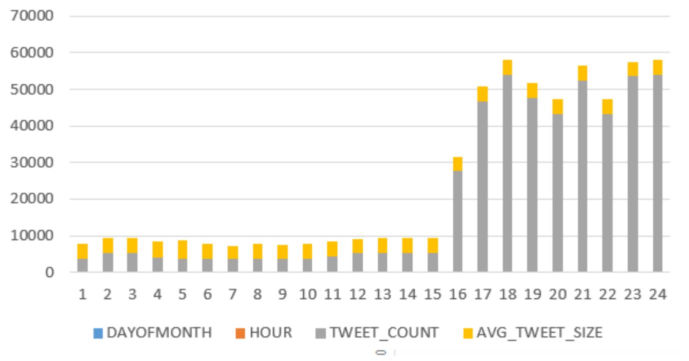

**Figure 3:** Average tweets per hour with average tweet size

## 3.6 Data cleansing, enrichment and transformation

For the experiment conducted Twitter accounts with less than 1,000 followers were considered and re-tweeted tweets were removed. From first observations it appears that this cleaned data set resulted in tweets that are more prone to have been sent by actual individuals.

In terms of enrichment below is an extract of the information added:

- Whether the user is part of the original tweet data-set (Set-1) retrieved from the Twitter stream (one of the tweets containing 'school' or 'homework' as a hash tag) or whether the user is a follower or friend of such a user (Set-2).

- Keeping track of the followers and friends of the users in the original data set.

## 3.7 Understanding the data gathered

As part of the experiment some initial variables were identified enabling an improved understanding of the social media data set gathered. Examples of these variables are described in Table 1.

**Table 1:** Initial variables identified for the experiment

| Variable | Description |
|---|---|
| R | average number of retweets per hour |
| U | total number of users |
| T | total number of tweets |
| AS | the number of users with the words ‘age’, ‘yr’ or ‘year’ in the status description |
| WO | the hashtags extrapolated from all tweets of Set-1 |
| TZ | the top time zones of all users |

Results for each of these variables were obtained and an extract of the results for some variables are described in Table 2.

**Table 2:** Initial results for the experiment

| Variable | Description | |
|---|---|---|
| R | on average 14,461 tweets were retweets out of the original average of 46,891 tweets per hour | |
| U | 2,686 users | |
| T | 265,535 tweets | |
| AS | 178 users out of 2,686 had the words ′age′, ′yr′ or ′year′ in their status description | |
| WO | It seems that ′nth grade′ and ′nth birthday′ are very common as shown by the word cloud in Fig. 4 below | |
| TZ | the top 3 entries: | |
| | Timezone | Count |
| | Pacific Time (US & Canada) | 26,450 |
| | Eastern Time (US & Canada) | 24,774 |
| | Central Time (US & Canada) | 18,384 |

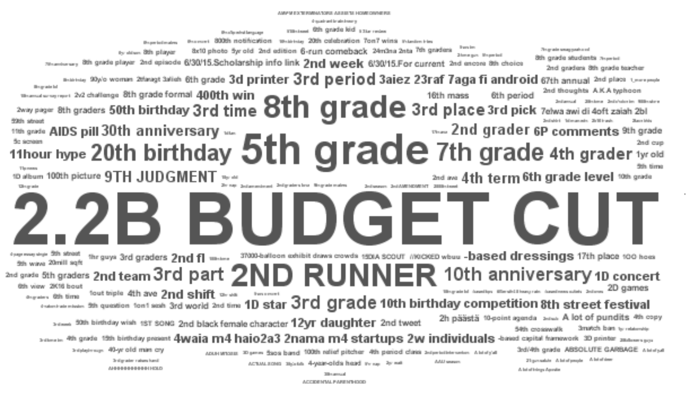

**Figure 4:** A word cloud of all tagged entries

**Initial insights from observing the data gathered**
Based on the results from interrogating the initial identified set of variables some interesting information has already presented itself towards being considered for inclusion in creating an Identity Deception Indicator (IDI).

- About 10 percent of all tweets mentioned the words 'yr', 'year' or 'age'. This is worth investigating to understand if this could be used as some form of age indicator.
- Only 25 percent of tweets were retweets. This seems a good indicator that the sample set is actual personal i.e. individual, Twitter accounts.

## 4 Conclusion

The protection of minors are still lacking in many aspects on the Internet and even more so with big data platforms like social media. This paper presents an initial attempt towards the early detection of identity deception so to protect minors on social media platforms. It is envisaged that the experiment, as discussed in this paper, together with its future expansions can assist authorities to pro-actively monitor social media feeds and identify potential online personas who are not who they pose to be.

## Acknowledgement

The support of "The HPI Future SOC" research Lab in Potsdam (Germany) is acknowledged for making available powerful infrastructure to conduct the research activities as presented in this paper.

## References

[1] M. Ali-ud-din Khan, M. F. Uddin, and N. Gupta. “Seven V’s of Big Data understanding Big Data to extract value”. In: *American Society for Engineering Education (ASEE Zone 1), 2014 Zone 1 Conference of the*. IEEE, pages 1–5.

[2] J. S. Alowibdi, U. A. Buy, S. Y. Philip, S. Ghani, and M. Mokbel. “Deception detection in Twitter”. In: *Social Network Analysis and Mining* 5.1 (2015), pages 1–16. ISSN: 1869-5450.

[3] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. Netto, and R. Buyya. “Big Data computing and clouds: Trends and future directions”. In: *Journal of Parallel and Distributed Computing* 79 (2015), pages 3–15.

[4] S. S. Baveja, A. D. Sarma, and N. Dalvi. *Determining trustworthiness and compatibility of a person*. Generic. 2015.

[5] D. Bogdanova, P. Rosso, and T. Solorio. “Exploring high-level features for detecting cyberpedophilia”. In: *Computer Speech & Language* 28.1 (2014), pages 108–120.

[6] L. Cai and Y. Zhu. “The Challenges of Data Quality and Data Quality Assessment in the Big Data Era”. In: *Data Science Journal* 14 (2015), page 2.

[7] L. Dedkova. “Stranger Is Not Always Danger: The Myth and Reality of Meetings with Online Strangers”. In: *LIVING IN THE DIGITAL AGE* (2015), page 78.

[8] A. O. Durahim and M. Coşkun. “# iamhappybecause: Gross National Happiness through Twitter analysis and big data”. In: *Technological Forecasting and Social Change* 99 (2015), pages 92–105.

[9] R. Han, Z. Jia, W. Gao, X. Tian, and L. Wang. “Benchmarking Big Data Systems: State-of-the-Art and Future Directions”. In: *arXiv preprint arXiv:1506.01494* (2015).

[10] E. Hargittai, J. Schultz, and J. Palfrey. “Why parents help their children lie to Facebook about age: Unintended consequences of the Children’s Online Privacy Protection Act”. In: *First Monday* 16.11 (2011).

[11] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan. “The rise of “big data” on cloud computing: review and open research issues”. In: *Information Systems* 47 (2015), pages 98–115.

[12] M. Y. Herring. *Social Media and the Good Life: Do They Connect?* McFarland, 2015.

[13] R. Kannadasan, R. Shaikh, and P. Parkhi. “Survey on big data technologies”. In: *International Journal of Advances in Engineering Research* Vol. No. 3 Issue No. III (2013).

[14] S. Kierkegaard. “Cybering, online grooming and ageplay”. In: *Computer Law & Security Review* 24.1 (2008), pages 41–55.

[15] I. Liccardi, M. Bulger, H. Abelson, D. J. Weitzner, and W. Mackay. "Can apps play by the COPPA Rules?" In: *Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference on*. IEEE, pages 1–9.

[16] S. Livingstone, G. Mascheroni, K. Ólafsson, and L. Haddon. "Children's online risks and opportunities: comparative findings from EU Kids Online and Net Children Go Mobile". In: (2014).

[17] F. Morstatter, J. Pfeffer, H. Liu, and K. M. Carley. "Is the Sample Good Enough? Comparing Data from Twitter's Streaming API with Twitter's Firehose". In: *ICWSM*.

[18] K. Moyle. "Filtering children's access to the internet at school". In: *ICICTE 2012 Proceedings* (2012).

[19] G. S. O'Keeffe and K. Clarke-Pearson. "The impact of social media on children, adolescents, and families". In: *Pediatrics* 127.4 (2011), pages 800–804.

[20] C. Peersman, W. Daelemans, and L. Van Vaerenbergh. "Predicting age and gender in online social networks". In: *Proceedings of the 3rd international workshop on Search and mining user-generated contents*. ACM, pages 37–44.

[21] F. Provost and T. Fawcett. *Data Science for Business: What you need to know about data mining and data-analytic thinking*. O'Reilly Media, Inc., 2013.

[22] H. A. Schwartz, J. C. Eichstaedt, M. L. Kern, L. Dziurzynski, S. M. Ramones, M. Agrawal, A. Shah, M. Kosinski, D. Stillwell, and M. E. Seligman. "Personality, gender, and age in the language of social media: The open-vocabulary approach". In: *PloS one* 8.9 (2013).

[23] L. Sloan, J. Morgan, P. Burnap, and M. Williams. "Who tweets? Deriving the demographic characteristics of age, occupation and social class from twitter user meta-data". In: *PloS one* 10.3 (2015).

[24] R. Von Solms and J. Van Niekerk. "From information security to cyber security". In: *Computers & Security* 38 (2013), pages 97–102.

[25] Wikipedia. *Identity Score*. Encyclopedia. 2015.

[26] R. Williams. "Children using social networks underage exposes them to danger". In: *The telegraph* (2014).

# A Scalable Query Dispatcher for Hyrise-R

Jan Lindemann, Stefan Klauck, David Schwalb

Enterprise Platform and Integration Concepts
Hasso Plattner Institute
jan.lindemann@student.hpi.de,stefan.klauck@hpi.de,david.schwalb@hpi.de

While single machines can handle the transactional database workload of most companies, the increasing analytical load will push them to their limit. For this reason, we extended the open source in-memory database Hyrise with the capability to form a database cluster for scalability and increased availability. This scale out and hot standby version is called Hyrise-R. It implements lazy master replication and has been shown to be well suited for mixed workloads as they exist in enterprise applications.

In this paper we present our extension of Hyrise-R: a query dispatcher, which works fully transparently and implements an enhanced query distribution algorithm. The new distribution algorithm improves load balancing and prioritizes write requests for higher transaction throughput. In addition, we discuss our work in progress and planned activities for Hyrise-R.

## 1 Introduction

Database workloads of companies can be classified into online transactional processing (OLTP) and online analytical processing (OLAP). Transactional workloads consist of write and read queries that access only a few tuples. On the other side, queries in analytical workloads access entire columns, e.g., for filtering, joins or aggregations. Both types have in common that read queries are the major part of the workload. Krüger et al. stated that 80 percent of the queries in OLTP workloads are read queries [6]. In OLAP workloads the ratio of read queries is even higher (90 percent). Since read requests do not change the data, multiple read requests can be executed concurrently without conflicts. Furthermore, the performance of the database can be increased with a scale out, e.g., a replication on multiple nodes. As a result, read queries can be spread over the different database instances and therefore more queries can be executed at the same time. Besides the increased throughput of the database, using multiple instances is beneficial for the system's resilience. If a database is replicated, every instance stores the same data. In case of an instance failure, another one can take over the work.

In-memory databases, like SAP HANA [9, 10], HyPer [5] or Hyrise [4, 3], enable a fast execution of mixed workloads, consisting of transactional and analytical queries, with a single database. This allows running analytical queries on up-to-date data which paves the way for new applications. In order to keep the database response times fast for the increasing workload, the database capacity has to be increased. To achieve this for the open source database Hyrise, we presented the implementation of lazy master replication called Hyrise-R [12]. Our previous work focused on the propagation of updates from the master node to the replicas in order to keep them up-to-date. This paper presents the progress on Hyrise-R, especially it contributes:

- The design of a configurable dispatcher to distribute the queries transparently among the cluster instances and its implementation for Hyrise-R.
- A detailed description of work in progress and next steps for Hyrise-R.

The presented implementations are publicly available as open source[1].

The next section gives an overview of Hyrise. Section 2 presents how Hyrise was extended to form a database cluster. Following we describe how query dispatching works in Hyrise-R. Section 5 presents our work in progress and next steps. Finally Section 6 concludes this paper.

## 2 Hyrise

Hyrise is an in-memory research database, developed at the Hasso Plattner Institute. By exploiting a main delta architecture, it is well suited for mixed workloads. Tuples in the main partition are thereby stored dictionary compressed with a sorted dictionary. This allows efficient vector scanning and supports optimized range queries in analytical workloads. New tuples are inserted in the write optimized delta partition. Using an unsorted dictionary is a trade-off for better write and reasonable read performance. The periodic merge process moves tuples from the delta to the main partition [6]. Hyrise supports a flexible hybrid table layout, allowing to store attributes corresponding to their access patterns [4, 3]. A columnar arrangement is well-suited for attributes which are often accessed sequentially, e.g., via scans, joins or aggregations. On the other side, attributes accessed in OLTP-style queries, e.g., projections of few tuples, can be stored in a row-wise manner. Hyrise exploits an insert only approach and multi-version concurrency control with snapshot isolation as default isolation level [11]. That is why Hyrise can process writes without delaying read queries.

## 3 Hyrise-R

Hyrise-R is a replication extension for the in-memory database Hyrise [12]. Figure 1 shows the architecture of Hyrise-R, comprising a query dispatcher and the database cluster. The Hyrise-R cluster consists of one master database instance, the primary node, and an arbitrary number of replica instances. Users send their requests to the dispatcher. Write requests are forwarded to the primary node. Read requests

[1]https://github.com/hyrise.

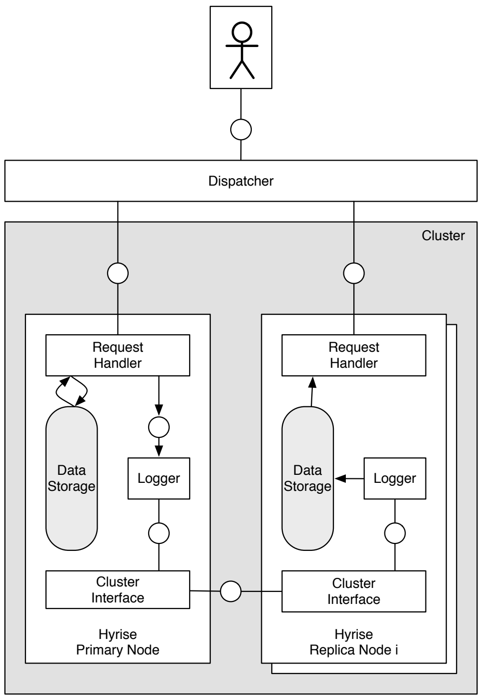

**Figure 1:** Architecture of Hyrise-R with the dispatcher and cluster

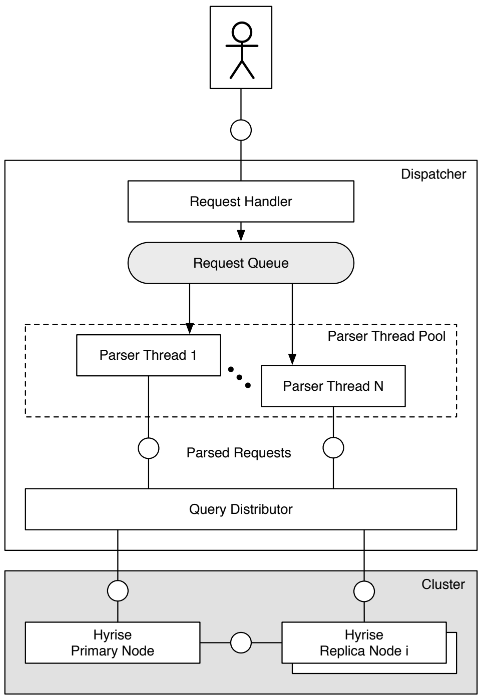

**Figure 2:** Architecture of the query dispatcher within Hyrise-R

are distributed among all cluster instances including the master in a round-robin manner.

To keep the replica nodes up-to-date, the primary node propagates changes to the replicas. Therefore, we added the Cluster Interface to the Hyrise core. It sends dictionary compressed logging information to the replica nodes. The replicas store the log entries and apply them to their table data. In order to reduce the number of exchanged messages, the changes are collected and transmitted in batches. This update process is called lazy replication [2]. To detect node failures, a heartbeat protocol is implemented. The primary node sends heartbeats to the replicas which have to acknowledge the reception. If the replicas do not receive a heartbeat in a certain interval, the next replica will take over the position as master instance. The new primary node will inform the dispatcher that the old master failed and that it takes over the work as primary node.

The focus of our previous work was the communication between cluster instances, i.e., the necessary modifications within Hyrise to support lazy master replication. To distinguish read and write queries, we used different endpoints (URLs) so that the dispatcher did not work completely transparently. Furthermore, the dispatcher distributed queries with the fixed distribution algorithm round-robin with no prioritization of writes. Reads would still be sent to the master node in write intensive workloads, which results in a heavier load for the master node compared to the replicas. In general, round-robin query distribution does not take into account:

- Different node hardware.
- Necessary compute and storage resources in order to calculate a query result.

In order to overcome these shortcomings, we refactored and extended the query dispatcher.

## 4 Dispatcher

The query dispatcher has the task to distribute incoming requests among the cluster instances. It has the same query interface as Hyrise so that it works transparently to the user. The dispatcher maintains a list of database hosts with a single primary node. We encapsulate this information in a configuration file, which is passed to the dispatcher on start-up. Besides the information about the database instances, i.e., their network address and start parameters, the file contains the dispatcher settings such as the query distribution algorithm and number of threads to use. Figure 2 shows an overview of the dispatcher as part of Hyrise-R. Query dispatching consists of two major steps: the parsing and distribution according to the request type.

First a request handler stores incoming requests, i.e., Hyrise JSON queries, in a request queue. Parser threads of a parser thread pool take the requests out of the queue in order to parse the JSON query plan. In the current implementation of the parser we distinguish between three cases:

- Read queries.
- Write queries, transactions with writes and procedures.
- Table loads.

The parser uses a blacklist approach to classify the request type. Therefore it maintains a list with all data manipulation operations. The parser scans the operators of a query and searches for data manipulation operations. Since the parser does not know whether a procedure will alter the data or not, each procedure will be executed on the primary node. Alternatively the procedures could declare their query type. The third request type is a table load, used to create and fill Hyrise tables. Since every node in the cluster has to store the same data, every instance must load the table. This is a special characteristic of a load query that distinguishes it from the other two types: a load query has to be distributed to all cluster nodes.

The parsed requests are passed to the query distributor. We implemented two distribution algorithms so far: a round robin distribution and so-called stream approach.

## 4.1 Round-Robin Approach

The round-robin approach distributes incoming read requests in circular order to the cluster instances. In order to determine the instance for the next read request, the parser uses a read counter. The id of the next instance is the current value of the counter modulo the number of cluster instances. After reading the value of the counter, it is incremented by one. In this way the read requests get assigned in circular order to the database instances. Write requests and procedures are always assigned to the master node. All requests are stored in the same queue (a queue per host implementation could also be used), together with the assigned host id and the socket with the connection to the requesting client.

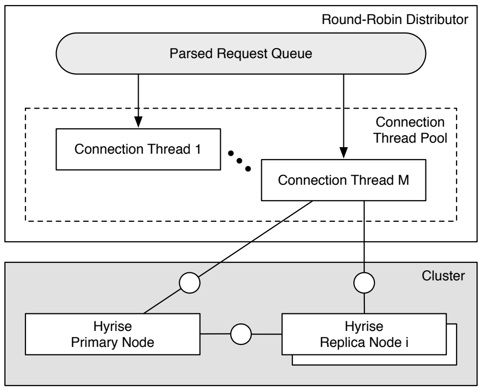

**Figure 3:** Round-robin distributor

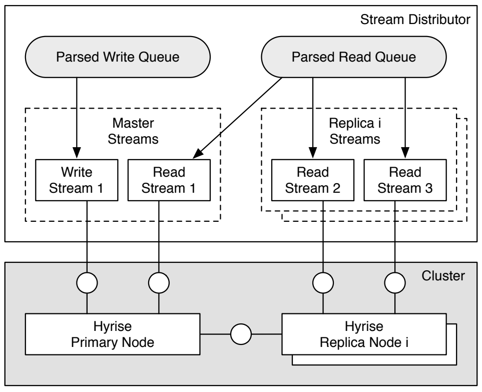

**Figure 4:** Stream distributor

Connection threads propagate the requests to the assigned database instances and respond to the client. Like depicted in Figure 3, the round-robin approach implements a thread pool containing these connection threads. The threads are created when the dispatcher starts and wait for tuples with parsed requests pushed into the parsed request queue.

## 4.2 Stream Approach

The second algorithm uses a fixed number of connections per cluster instance. The idea of the algorithm is to execute write queries preferentially and to increase the transactional throughput in this way. In addition, it supports heterogeneous clusters by assigning different numbers of connection threads to the cluster instances.

The algorithm separates read and write requests and stores them into one of two different queues. One queue contains all parsed read requests, whereas the other one contains all parsed write requests. In contrast to the round-robin approach, this approach does not use a thread pool shared by all cluster instances but assigns a fixed number of threads to each cluster instance (see Figure 4). These threads are called streams, listen on one of the queues and process requests sequentially.

Streams belonging to one of the replicas can only listen and take requests from the read request queue. The streams that are bound to the master can listen either on the read queue or on the write queue. However, at least one master stream has to listen on the write queue. If a stream gets a request from the queue it is listening on, it sends the query to its database instance and returns the result to the client.

## 5 Discussion and Future Work

This paper presents an extended query dispatcher for Hyrise-R, which works transparently to the user and supports various query distribution algorithms. The stream approach overcomes the disadvantages of a round-robin distribution as it can prioritize write queries on the master node and handle imbalances in the cluster hardware by assigning more query streams to more powerful cluster nodes. This will increase the query performance for our planned evaluation.

For performance measurements of databases, Cole et al. presented the mixed workload CH-benCHmark [1]. It combines the transactional TPC-C and analytical TPC-H benchmark. The basis of the database schema is the transactional schema of the TPC-C benchmark, extended by tables only used in the TPC-H benchmark. The transactional queries are taken from the TPC-C benchmark. The analytical queries are modified queries of the TPC-H benchmark which are adapted according to the combined schema. Our primary goal and work in progress is a comprehensive performance analysis of Hyrise-R. We plan to base our evaluation on the CH-benCHmark. TPC-C benchmark is already implemented with stored procedures in Hyrise. For the TPC-H benchmark, some necessary operators are not yet implemented in Hyrise.

Besides a performance analysis of the database cluster, we plan to compare various distribution algorithms and implement more sophisticated approaches. The architecture of the described dispatcher allows extending the functionality of existing query distributors and creating new distribution algorithms with low effort. So far we focused on homogeneous cluster instances. But also scenarios with heterogeneous machines are possible, i.e., nodes with different hardware resources or different runtime configurations, e.g., indices. One of our ideas is to setup the cluster instances with different indices. The dispatcher can exploit the parsed query and the knowledge about cluster instances to distribute the queries to appropriate nodes according to the accessed columns.

Further, we plan to increase the resilience of the database cluster. A potential problem is a failure of the master node and a loss of logs which have not been replicated yet. As future work we plan to discuss possibilities to achieve k-safety, e.g., distributed logs. Moreover, the dispatcher is still a single point of failure. Even though the dispatcher is relatively simple and not as error-prone as complex software like a database, a failure of the dispatcher would cause a failure of the whole database cluster. David et al. proposes a cluster of dispatchers with failure detection as a possible solution [12]. In case that one dispatcher fails, another could take over the work.

The implementation of elasticity, the capability to extend and shrink the database cluster depending on the current workload, is a further goal for Hyrise-R.

Hyrise-R is similar to ScyPer [8], a scale out version of HyPer [5]. The master node sends redo logs to the replicas using multicasts. A coordinator process at the primary node distributes read queries among the secondary HyPer nodes [7].

## 6 Conclusion

This paper presents an advanced query dispatcher for Hyrise-R, the scale out version of the in-memory database Hyrise. The configurable and extensible query dispatcher consists of two major parts, a request parser and distributor. The query distributor uses the parsed query information to propagate the query to an appropriate cluster instance. It uses a blacklist approach to distinguish between read, write and table load requests. Advanced distribution algorithms can exploit knowledge about the cluster instances, e.g., existing indices to forward queries to the best suitable node. We discussed how the implemented stream distribution algorithm improves load balancing and increases the transaction throughput. For future work we plan to demonstrate the capabilities of Hyrise-R in a performance evaluation.

## Acknowledgment

Stefan Klauck has received funding from the European Union's Horizon 2020 research and innovation program 2014–2018 under grant agreement No. 644866 (SSICLOPS). This document reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

## References

[1] R. Cole, F. Funke, L. Giakoumakis, W. Guy, A. Kemper, S. Krompass, H. Kuno, R. Nambiar, T. Neumann, M. Poess, K.-U. Sattler, M. Seibold, E. Simon, and F. Waas. "The Mixed Workload CH-benCHmark". In: *Proceedings of the Fourth International Workshop on Testing Database Systems*. 2011.

[2] J. Gray, P. Helland, P. O'Neil, and D. Shasha. "The Dangers of Replication and a Solution". In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. SIGMOD '96. 1996.

[3] M. Grund, P. Cudre-Mauroux, J. Krüger, S. Madden, and H. Plattner. "An overview of HYRISE–A Main Memory Hybrid Storage Engine". In: *IEEE Data Engineering Bulletin* (2012).

[4] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden. "HYRISE: A Main Memory Hybrid Storage Engine". In: *Proc. VLDB Endow.* (2010).

[5] A. Kemper and T. Neumann. "HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots". In: *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. 2011.

[6] J. Krüger, C. Kim, M. Grund, N. Satish, D. Schwalb, J. Chhugani, H. Plattner, P. Dubey, and A. Zeier. "Fast Updates on Read-optimized Databases Using Multi-core CPUs". In: *Proc. VLDB Endow.* (2011).

[7] T. Mühlbauer, W. Rödiger, A. Reiser, A. Kemper, and T. Neumann. "ScyPer: A Hybrid OLTP&OLAP Distributed Main Memory Database System for Scalable Real-Time Analytics". In: *BTW*. 2013.

[8] T. Mühlbauer, W. Rödiger, A. Reiser, A. Kemper, and T. Neumann. "ScyPer: Elastic OLAP Throughput on Transactional Data". In: *Proceedings of the Second Workshop on Data Analytics in the Cloud*. DanaC '13. 2013.

[9] H. Plattner. "A Common Database Approach for OLTP and OLAP Using an In-memory Column Database". In: *SIGMOD* (2009).

[10] H. Plattner. "The Impact of Columnar In-memory Databases on Enterprise Systems: Implications of Eliminating Transaction-maintained Aggregates". In: *Proc. VLDB Endow.* (2014).

[11] D. Schwalb, M. Faust, J. Wust, M. Grund, and H. Plattner. "Efficient Transaction Processing for Hyrise in Mixed Workload Environments". In: *IMDM in conjunction with VLDB*. 2014.

[12] D. Schwalb, J. Kossmann, M. Faust, S. Klauck, M. Uflacker, and H. Plattner. "Hyrise-R: Scale-out and Hot-Standby Through Lazy Master Replication for Enterprise Applications". In: *Proceedings of the 3rd VLDB Workshop on In-Memory Data Mangement and Analytics*. 2015.

# A Survey of Security-Aware Approaches for Cloud-Based Storage and Processing Technologies

Max Plauth, Felix Eberhardt, Frank Feinbube and Andreas Polze

Operating Systems and Middleware Group
Hasso Plattner Institute for Software Systems Engineering
firstname.lastname@hpi.de

In the Gartner hype cycle, cloud computing is a paradigm that has crossed the peak of inflated expectations but also has overcome the worst part of the trough of disillusionment. While the advantages of cloud computing are the best qualification for traversing the slope enlightenment, security concerns are still a major hindrance that prevent full adoption of cloud services across all conceivable user groups and use cases. With the goal of building a solid foundation for future research efforts, this paper provides a body of knowledge about a choice of upcoming research opportunities that focus on different strategies for improving the security level of cloud-based storage and processing technologies.

## 1 Introduction

Providing low total cost of ownership, high degrees of scalability and ubiquitous access, cloud computing offers a compelling list of favorable features to both businesses and consumers. At the same time, these positive qualities also come with the less favorable drawback, that guaranteeing data confidentiality in cloud-based storage and processing services still remains an insufficiently tackled problem. As a consequence, many companies and public institutions are still refraining from moving storage or processing tasks into the domain of cloud computing. While this reluctance might be appropriate for few, highly sensitive use-cases, it poses the risk of an economic disadvantage in many other scenarios.

This paper provides an overview about the current state of the art in security-aware approaches for cloud-based storage and processing technologies. Since there are numerous ways to approach the topic, a large variety of potential starting points is presented. The goal is to provide a solid body of knowledge, which will be used as a foundation upon which novel security mechanisms can be identified and studied in the future. In the ensuing section, we present a selected list of preceding contributions to the field of security research in the context of cloud computing. Afterwards, a comprehensive review of the state of the art is provided to form a body of knowledge.

## 2 Preceding contributions

In the last couple of years, several aspects relevant to security-aware approaches for cloud-based storage and processing technologies have been researched at our research group. Among these aspects are technologies such as threshold cryptography, trust-based access control, virtual machine introspection and searchable encryption. Since our ongoing research efforts build up on top of the insights gained in these preceding contributions, a brief overview is provided.

### 2.1 Threshold Cryptography

In a widely distributed environment, traditional authorization services represent a single-point of failure: If the service is unavailable, the encrypted data cannot be accessed by any party. In distributed setups, simple replication mechanisms can be considered a security threat, since attackers can gain full control as soon as a single node has been compromised. In order to eliminate this weakness, the general approach presented by Neuhaus et al. (2012) [29] employs the concept of Fragmentation-Redundancy-Scattering [9]: Confidential information is broken up into insignificant pieces which can be distributed over several network nodes.

The contribution of Neuhaus et al. (2012) [29] is the design of a distributed authorization service. A system architecture has been presented that enables fine-grained access control on data stored in a distributed system. In order to maintain privacy in the presence of compromised parties, a threshold encryption scheme has been applied in order to limit the power of a single authorization service instance.

### 2.2 Trust-Based Access Control

The Operating System and Middleware Group operates a web platform called *InstantLab* [28, 27]. The purpose of the platform is to provide operating system experiments for student exercises in the undergraduate curriculum. Virtualization technology is used to provide pre-packaged experiments, which can be conducted through a terminal session in the browser. Thus far, massive open online-courses (MOOCs) have not been well suited for hands-on experiments, since assignments have been non-interactive. The main goal of *InstantLab* is to provide more interactive assignments and enable iterative test-and-improve software development cycles as well as observational assignments.

Providing a platform that enables a large audience to perform live software experiments creates several challenges regarding the security of such a platform. Malicious users might abuse resources for other means than the intended software experiments. In order to detect misuse of the provided resources, virtual machine introspection is applied. Furthermore, *InstantLab* [28, 27] demonstrates how automatic resource management is enabled by trust-based access control schemes. The purpose of trust-based access control is to restrict user access to resource intensive experiments. The approach implemented in *InstantLab* [28, 27] calculates a user's trust level based on his/her previous behavior.

## 2.3 Virtual Machine Introspection

In the age of cloud computing and virtualization, virtual machine introspection provides the means to inspect the state of virtual machines through a hypervisor without the risk of contaminating its state. Inspection capabilities are useful for a wide range of use case scenarios, ranging from forensics to more harmless cases such as making sure a tenant is not violating against the terms of use of the provider.

The work of Westphal et al. (2014) [36] contributes to the field of virtual machine introspection by providing a monitoring language called VMI-PL. Using this language, users can specify which information should be obtained from a virtual machine. Unlike competing approaches like libVMI [19] and VProbes [35], VMI-PL does not limit users to hardware level metrics, but it also provides operating system level information such as running processes and other operating system events. Furthermore, the language can also be used to monitor data streams such as network traffic or user interaction.

## 2.4 Searchable Encryption

For many use cases, efficient and secure data sharing mechanisms are crucial, especially in distributed scenarios where multiple parties have to access the same data repositories from arbitrary locations. In such scenarios, the scalability of cloud computing makes resources simple to provision and to extend. However, when it comes to storing sensitive data in cloud-hosted data repositories, data confidentiality is still a major issue that discourages the use of cloud resources in sensitive scenarios. While traditional encryption can be used to protect the privacy of data, it also limits the set of operations that can be performed efficiently on encrypted data, such as search. Encryption schemes which allow the execution of arbitrary operations on encrypted data are still utopian. However, searchable encryption schemes exist that enable keyword-based search without the disclosure of keywords.

Neuhaus et al. (2015) [26] studied the practical applicability of searchable encryption for data archives in the cloud. For their evaluation, an implementation of Goh's searchable encryption scheme [12] was embedded into the document-based database MongoDB. With the encryption scheme in place, benchmarks revealed that the overhead for insertions is negligible compared to an unencrypted mode of operation. Search queries on the other hand come with a considerable overhead, since Goh's scheme [12] mandates a linear dependency between the complexity of search operations and the number of documents. However, the processing time of encrypted queries should be in acceptable orders of magnitude for interactive use cases where the increased security is mandatory.

# 3 State of the Art

In the context of cloud computing, the field of work related to security-aware approaches for storage and processing technologies comprises a wide range of diverse directions. In the consequent part of this document, the state of the art is presented for a selection of differentiated topics. First, projects are highlighted which provide best practices for increasing security. Afterwards, new trends in virtualization strategies are outlined, followed by a brief introduction to novel hardware security mechanisms. Finally, the security aspects of providing coprocessor resources in virtual machines is illustrated.

## 3.1 New trends in virtualization strategies

Virtualization still remains as one of the main technological pillars of cloud computing. The main reason for this key role is that it enables high degrees of resource utilization and flexibility. Today, the most common approach for virtualization resorts to low-level hypervisors like *Xen* or *KVM* that employ hardware assisted virtualization in order to run regular guest operating systems in a para-virtualized or fully virtualized fashion. Recently however, new virtualization approaches have gained momentum. While containerization approaches move the scope of virtualization to higher levels of the application stack, unikernels are working at the same level of abstraction as regular operating systems but at the same time change the operating system drastically. A comparison of the different approaches is illustrated in Figure 1.

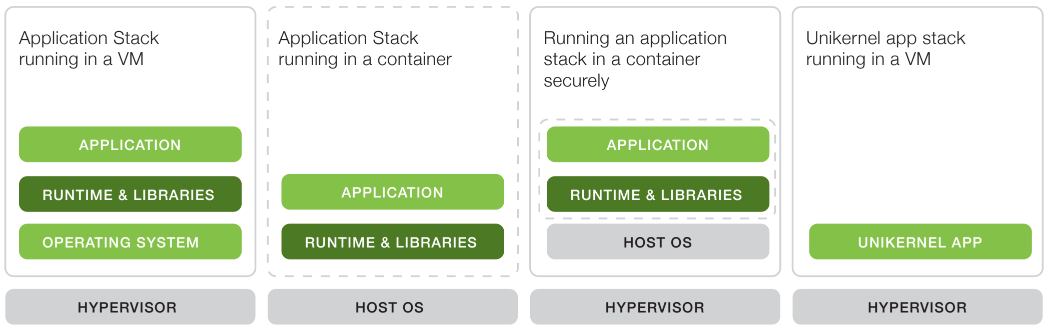

**Figure 1:** The virtual machine stack as well as both containerization approaches come with a significant amount of overhead. Unikernels aim at reducing the footprint of virtualized applications. Source: [38]

**Containers**
In contrast to hypervisor-level virtualization approaches, where an entire operating system instance is virtualized, containers belong to the class of operating-system-

level virtualization strategies that utilize multiple user-space instances in order to isolate tenants. The main goal of popular containerization implementations like Linux Containers and Docker is to reduce the memory footprint of hosted applications and to get rid of the overhead inherent to hypervisor-based virtualization. Recent studies demonstrate that the concept of containerization is able to outperform matured hypervisors in many use cases [33, 37, 10]. Regarding security aspects, most containerization approaches thus far rely on the operating system kernel to provide sufficient means of isolation between different containers. However, the LXD project aims at providing hardware-based security features to containers in order to provide isolation levels on par with hypervisor-level based virtualization.

**Unikernels**
Unikernels are a new approach to hypervisor-level virtualization. The core concept of unikernels is based on the idea of deploying applications by merging application code and a minimal operating system kernel into a single immutable virtual machine image that is run on top of a standard hypervisor [22, 20, 31]. Since unikernels intentionally do not support the concept of process isolation, no time-consuming context switches have to be performed. The general idea behind unikernel systems is not entirely new, as it builds up on top of the concept of *library operating systems* such as exokernel [8] or Nemesis [30]. The main difference to library operating systems is that unikernels only run on hypervisors and do not support bare metal deployments, whereas library operating systems are targeting physical hardware. Due to the necessity to support physical hardware, library operating systems struggled with compatibility issues and proper resource isolation among applications. Unikernels are solving these problems by using a hypervisor in order to abstract from physical hardware and to provide strict resource isolation between applications [20]. Currently, the two most popular unikernel implementations are OSv [17] and Mirage OS [21]. According to Madhavapeddy et al. [20], unikernels are able to outperform regular operating systems in the following aspects:

**Boot time** Unikernel systems are single purpose systems, meaning that they run only one application. Unnecessary overhead is stripped of by only linking libraries into a unikernel image which are required by the application. As a result, very fast boot times can be achieved. In their latest project *Jitsu: Just-In-Time Summoning of Unikernels* [3], Madhavapeddy et al. managed to achieve boot times in the order of 350ms on ARM CPUs and 30ms on x86 CPUs, which enables the possibility of dynamically bringing up virtual machines in response to network traffic.

**Image size** Since a unikernel system only contains the application and only the required functionality of the specialized operating system kernel, unikernel images are much smaller compared to traditional operating system images. The smaller binaries simplify management tasks like live-migration of running virtual machine instances.

**Security** By eliminating functionality which is not needed for the execution of an application inside a unikernel image, the attack surface of the system is reduced massively. Furthermore, the specialized operating system kernel of a unikernel image is usually written in the same high-level language as the application. The resulting absence of technology borders facilitates additional opportunities for code checking like static type checking and automated code checking. However, even if an attacker should manage to inject malicious code into a unikernel instance, it can only cause limited harm since no other application runs within the same image.

## 3.2 Hardware-based security mechanisms

**Trusted Execution Technology (TXT)**

The goal of Intel Trusted Execution Technology (TXT) [16] technology is that the user can verify if the operating system or its configuration was altered after the boot up. This requires a trusted platform module (TPM) which stores system indicators securely. The approach TXT is using is called dynamic root of trust measurement. For this methodology, the system can be brought into a clean state (SENTER instruction) after the firmware was loaded. In this approach as mentioned earlier only the operating system level software gets measured. These measurements can be compared with the original files or properties of the OS that have to be known beforehand.

**Software Guard Extensions (SGX)**

Sensitive tasks have to face an abundance of potential threats on both the software and the hardware level. On the hardware level, sensitive information such as encryption keys can be extracted from the systems main memory using DMA attacks or cold boot attacks. On the software level, the worst case has to be assumed and even the operating system has to be considered as a potential threat. While the concept of processes implements a high level of isolation between different applications, the elevated privileges of an operating system allow it to tamper with any process. These capabilities always pose a security threat, not just in the obvious case where the operating system might not be fully trusted. Even with a trusted operating system, there is always a certain risk that malicious code running in a separate process might gain elevated privileges. As soon as that happens, a malicious application can tamper with any process running on the system.

As a countermeasure to these threats, the Intel Software Guard Extensions (SGX) [25] introduced secure enclaves, which allow the safe execution of sensitive tasks even in untrustworthy environments. Enclaves are protected memory areas, which are encrypted and entirely isolated (see Figure 2). Even privileged code is not able to access the contents of an enclave. One process can even use multiple enclaves, which allows a high degree of flexibility. SGX does not require a trusted platform module (TPM), as the entire feature is implemented on the CPU. This level of integration reduces the list of trusted vendors to the CPU manufacturer and thus minimizes the number of potential attack vectors.

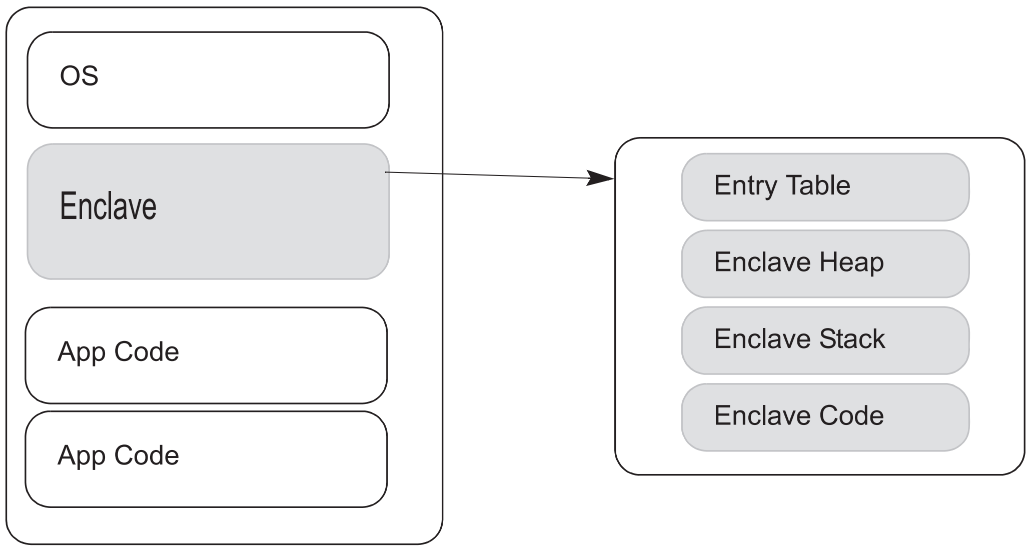

**Figure 2:** Secure enclaves provide an encrypted address space that is protected even from operating system access. Source: [15]

## 3.3 Virtualization of coprocessors resources

Coprocessors such as *Graphics Processing Units* (GPUs), *Field-Programmable Gate Arrays* (FPGAs) or Intel's *Many Integrated Core* (MIC) devices have become essential components in the *High Performance Computing* (HPC) field. Regarding the domain of cloud computing however, the utilization of coprocessors is not that well established. While there has been little demand for HPC-like applications on cloud resources in the past, the demand for running scientific applications on cloud computing infrastructure has increased [2]. Furthermore, moving compute-intensive applications to the cloud is becoming increasingly feasible [23] when it comes to CPU-based tasks. While several providers already offer cloud resources with integrated GPUs, their implementation is based on pass-through of native hardware. Assigning dedicated devices to each virtual machine results in high operational costs and decreased levels of flexibility. In such setups, virtual machines can neither be suspended nor migrated. Furthermore, the one-to-one mapping between pass-through devices and virtual machines prevent efficient utilization of coprocessors, which has a negative impact on cost effectiveness due to the high energy consumption of such hardware. Although projects exist that maximize resource utilization by providing unused GPU resources to other compute nodes [24] or that save energy by shutting down inactive compute nodes [18], a large gap between the capabilities of GPU and CPU virtualization still exists.

In the ensuing paragraphs, the state of the art of coprocessor virtualization is evaluated based on several characteristics. Most work deals with GPU compute devices, however the general techniques are applicable to other coprocessor classes as well. For desktop-based GPU virtualization, Dowty and Sugerman [4] define four characteristics that are to be considered: performance, fidelity, multiplexing and interposition. Regarding cloud-based virtualization, the aforementioned enumeration is missing isolation as an important characteristic. In order to establish coprocessors in cloud computing, one of the most crucial characteristics is that multiple tenants have to be properly isolated. Since the focus is set on coprocessors and thus compute-based capabilities instead of interactive graphics, fidelity can mostly be ignored for our use case. Last but not least, performance should not suffer severely from the virtualization overhead. However, without isolation, multiplexing and interposition capabilities, performance is worthless in the cloud computing use case.

**Isolation**
Thus far, isolation is only addressed by approaches that make use of mediated pass-through strategies like Intel GVT-g (formerly called gVirt) [34] and NVIDIA GRID. While the latter is a commercial closed-source implementation, the implementation details of GVT-g are publicly available as an open source project. In Intel's approach, each virtual machine runs the native graphics driver. In contrast to regular pass-through, mediated pass-through uses a trap-and-emulate mechanism is used to isolate virtual machine instances from each other. The main drawback is that the implementation of the mediated pass-through strategy has to be tailored to the specifications of each supported GPU, which again requires detailed knowledge about the GPU design. Overall, this approach is only feasible for the manufacturers of GPUs themselves.

With rCUDA [5, 7, 6], vCUDA [32], gVirtuS [11], GViM [13], VirtualCL [1] and VOCL [39], many approaches exist which are based on call forwarding. Originating from the field of High Performance Computing, the call forwarding approach uses a driver stub in the guest operating system which redirects the calls to a native device driver in the privileged domain. Since isolation is barely an issue in the HPC domain, none of the existing approaches implement isolation mechanisms.

**Multiplexing**
Sharing a single GPU among multiple virtual machines is possible for all aforementioned implementation strategies. In the faction of mediated pass-through implementations, both Intel GVT-g and NVIDIA GRID support multiplexing in order to serve multiple virtual machines with a single GPU. As for isolation, a trap-and-emulate mechanism in the hypervisor coordinates devices accesses from multiple virtual machines. On the side of call forwarding approaches, the implementation of multiplexing capabilities with low overhead is very a tough challenge. In the privileged domain, additional logic has to be implemented that schedules requests from different guests. So far, only vCUDA [32] provides such multiplexing mechanisms.

**Interposition**
While mediated pass-through approaches excel call forwarding strategies in both isolation and multiplexing, interposition is hard to achieve for mediated pass-through. Although an implementation is possible in theory [40], it is not feasible in practice as it is susceptible to the slightest variations on the hardware level. With vCUDA [32] and VOCL [39] on the other hand, multiple projects based on call forwarding exist that successfully implement interposition capabilities. Again, a piece of middleware is required in the hypervisor which carefully tracks the state of each virtual GPU instance. With such capabilities at hands, virtual machines can be suspended and even live-migrated to other virtual machine hosts.

## 3.4 Best practices for secure coding

Over the last years, several best practice collections and frameworks dealing with improving the security of information technology were established and maintained by companies and public authorities alike.

**Critical Security Controls**
The term "Security Fog of More" was established by Tony Sager, a chief technologist of the *Council on CyberSecurity*. He noticed that security professionals are confronted with a plethora of security products and services. These choices are influenced by compliance, regulations, frameworks and audits e.g. the "Security Fog of More". As a consequence, one of the main challenges today is making an educated choice. Sager wants to help security professionals by providing a framework for security choices called *Critical Security Controls* [3] (see Figure 3) that spans 20 different areas of IT security containing suggestions for each of these areas with a focus on scalability of the solutions.

**Open Web Application Security Project**
The Open Web Application Security Project (OWASP) is a non-profit organization founded 2004 with the goal of improving software security. The OWASP houses a wide range of security related projects centered around all aspects of software development driven by a large community of volunteers. We will provide a brief overview over a selection of the most popular OWASP projects:

**OWASP Developer Guide** The OWASP Developer Guide was the first project pursued by OWASP. In its latest revision, the guide describes general concepts about developing secure software without a focus on specific technologies. The guide covers topics such as Architecture, Design, Build Process and Configuration of secure software and is targeting developers as its target audience. The instructions can be used as additional guidelines for penetration testers as well.

**OWASP Testing Guide** The OWASP Testing Guide is a best practice collection for penetration testing of web applications and services. The guide covers the

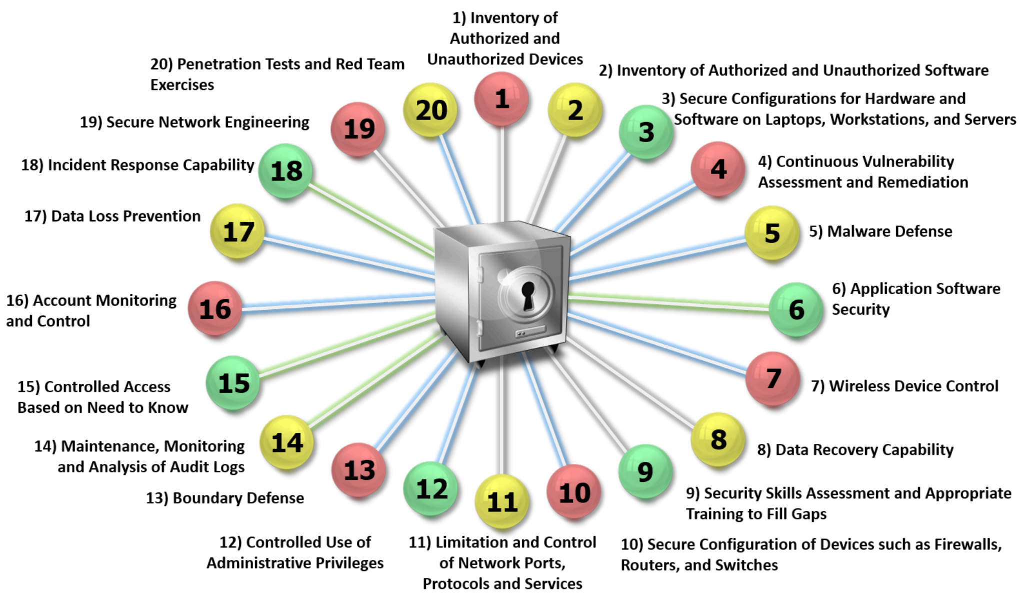
1) Inventory of Authorized and Unauthorized Devices
2) Inventory of Authorized and Unauthorized Software
3) Secure Configurations for Hardware and Software on Laptops, Workstations, and Servers
4) Continuous Vulnerability Assessment and Remediation
5) Malware Defense
6) Application Software Security
7) Wireless Device Control
8) Data Recovery Capability
9) Security Skills Assessment and Appropriate Training to Fill Gaps
10) Secure Configuration of Devices such as Firewalls, Routers, and Switches
11) Limitation and Control of Network Ports, Protocols and Services
12) Controlled Use of Administrative Privileges
13) Boundary Defense
14) Maintenance, Monitoring and Analysis of Audit Logs
15) Controlled Access Based on Need to Know
16) Account Monitoring and Control
17) Data Loss Prevention
18) Incident Response Capability
19) Secure Network Engineering
20) Penetration Tests and Red Team Exercises

**Figure 3:** The Critical Security Controls framework categorizes security threats in 20 classes. Source: [14]

software development process as well as testing approaches for different parts of web applications (e.g. Authentication, Encryption or Input Validation).

**OWASP Top 10** The OWASP Top 10 is a list maintained by security experts which contains the 10 most prevalent security flaws in web applications. The goal of this list to establish a security awareness in IT companies to prevent the occurrence of the most common vulnerabilities in their applications.

## 4 Discussion

The state of the art presented in the previous section has demonstrated that a vast variety of approaches exist that can be accommodated under the headline *security-aware approaches for cloud-based storage and processing technologies*. While soft approaches such as best practice collections are beneficial for everyday use, they are of limited use for technically oriented research prototypes. On the other end of the scale are hardware security features such as TXT and SGX.

The Software Guard Extensions is an interesting new feature that can be used to evaluate problems that require the execution of crucial code in untrustworthy requirements. However, it should be noted that until the day of writing, no commercially available processor implements the SGX feature. Moreover, it is even unclear when such a processor can be expected to become available. Under the bottom line, it seems like SGX provides various research opportunities, however the focus for near future projects should be shifted to different topics.

With virtualization being a key technology in cloud computing, it is important to keep an eye on new virtualization concepts. With the advent of containerization, a new approach to virtualization has surfaced that tries to minimize the performance overhead caused by an additional level of context switches. While containers have already achieved a certain prevalence rate, unikernels are a recent re-discovery of an old concept. Unikernels should be considered as a direct competition to containers, since they also address mitigation of virtualization overhead while maintaining a thorough level of isolation. Even though there is a certain risk that unikernels might be a fashionable trend, eventual benefits over traditional virtual machines and containerization approaches should be evaluated. With boot times of tens of milliseconds, the use of unikernels might enable new degrees of dynamic resource utilization and improved power management.

In the subject area of virtualization, server-based virtualization of coprocessor resources, most importantly *Graphics Processing Units* (GPUs), is another aspect that has been neglected in the past. High operational costs are caused by poorly utilized devices. Even though some approaches exist that allow resource multiplexing, the near absence of proper isolation has been a deal-breaker for the cloud computing scenario thus far.

# 5 Outlook

Recalling the topics presented and discussed in Sections 3 and 4, many approaches exist for providing increased levels of security in the use case of cloud computing. While best practices collections may be beneficial for everyday use, they are of limited use for technically oriented research interests. It seems as if technological improvements like the Software Guarded Extensions (SGX) are an interesting target for further research efforts. However, the uncertain date of availability of the technology enforces a postponed examination of the topic. Regarding new virtualization approaches, there is a certain risk that unikernels are a fashionable trend that might disappear rather sooner than later. However, the crucial role of virtualization in cloud computing suggests that unikernels and containers should be evaluated more thoroughly. From a functional perspective, these new virtualization approaches do have the potential to improve both security aspects as well as performance. Regarding the non-function side of the topic, unikernels might enable us to improve both dynamic resource utilization and power management strategies. Last but not least, employing coprocessor resources in cloud computing is a topic that requires extensive research efforts. In order to move from dedicated devices to truly shared resources, security is a major concern that has not been solved yet. Lightweight isolation mechanisms have to be researched that provide tight levels of isolation while inducing bearable levels of overhead compared to native hardware.

## Acknowledgement

This paper has received funding from the European Union's Horizon 2020 research and innovation programme 2014–2018 under grant agreement No. 644866.

## Disclaimer

This paper reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

## References

[1] A. Barak and A. Shiloh. *The VirtualCL (VCL) Cluster Platform*.

[2] S. Benedict. "Performance issues and performance analysis tools for HPC cloud applications: a survey". In: *Computing* 95.2 (2013), pages 89–108.

[3] Council on CyberSecurity. *The Critical Security Controls for Effective Cyber Defense Version 5.0*. Technical report. https://www.sans.org/, 2014.

[4] M. Dowty and J. Sugerman. "GPU virtualization on VMware's hosted I/O architecture". In: *ACM SIGOPS Operating Systems Review* 43.3 (2009), pages 73–82.

[5] J. Duato, F. D. Igual, R. Mayo, A. J. Peña, E. S. Quintana-Ortí, and F. Silla. "An Efficient Implementation of GPU Virtualization in High Performance Clusters". In: *Euro-Par 2009 – Parallel Processing Workshops*. Edited by H.-X. Lin, M. Alexander, M. Forsell, A. Knüpfer, R. Prodan, L. Sousa, and A. Streit. Volume 6043. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pages 385–394. DOI: 10.1007/978-3-642-14122-5.

[6] J. Duato, A. J. Pena, F. Silla, J. C. Fernandez, R. Mayo, and E. S. Quintana-Orti. "Enabling CUDA acceleration within virtual machines using rCUDA". English. In: *2011 18th International Conference on High Performance Computing*. IEEE, Dec. 2011, pages 1–10. DOI: 10.1109/HiPC.2011.6152718.

[7] J. Duato, A. J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Orti. "rCUDA: Reducing the number of GPU-based accelerators in high performance clusters". English. In: *2010 International Conference on High Performance Computing & Simulation*. IEEE, June 2010, pages 224–231. DOI: 10.1109/HPCS.2010.5547126.

[8] D. R. Engler, M. F. Kaashoek, et al. *Exokernel: An operating system architecture for application-level resource management*. Volume 29. 5. ACM, 1995.

[9] J.-C. Fabre, Y. Deswarte, and B. Randell. *Designing secure and reliable applications using fragmentation-redundancy-scattering: an object-oriented approach*. Springer, 1994.

[10] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. *An updated performance comparison of virtual machines and linux containers*. Technical report. 2014, page 32.

[11] G. Giunta, R. Montella, G. Agrillo, and G. Coviello. "A GPGPU transparent virtualization component for high performance computing clouds". In: *Euro-Par 2010-Parallel Processing*. Springer, 2010, pages 379–391.

[12] E.-J. Goh et al. "Secure Indexes." In: *IACR Cryptology ePrint Archive* 2003 (2003), page 216.

[13] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, and P. Ranganathan. "GViM: GPU-accelerated Virtual Machines Vishakha". In: *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing - HPCVirt '09*. New York, New York, USA: ACM Press, Mar. 2009, pages 17–24. DOI: 10.1145/1519138.1519141.

[14] F. T. Insider. *Continuous Diagnostics and Mitigation Addresses "Foundational" Issues Identified by SANS. http://www.federaltechnologyinsider.com/cdm-addresses-foundational-issues-identified-sans/.* May 2014.

[15] Intel Corporation. *Intel© Software Guard Extensions Programming Reference*. Technical report. Oct. 2014.

[16] Intel©Corporation. *Intel©Trusted Execution Technology White Paper*. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/trusted-execution-technology-security-paper.pdf. Online, Accessed 31.07.2015.

[17] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har'El, D. Marti, and V. Zolotarov. "OSv–Optimizing the operating system for virtual machines". In: *2014 usenix annual technical conference (usenix atc 14)*. Volume 1. USENIX Association. 2014, pages 61–72.

[18] P. Lama, Y. Li, A. M. Aji, P. Balaji, J. Dinan, S. Xiao, Y. Zhang, W.-c. Feng, R. Thakur, and X. Zhou. "pVOCL: Power-Aware Dynamic Placement and Migration in Virtualized GPU Environments". English. In: *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, July 2013, pages 145–154. DOI: 10.1109/ICDCS.2013.51.

[19] LibVMI Project. *LibVMI*. http://libvmi.com. Accessed: 2015-07-17.

[20] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. "Unikernels: Library operating systems for the cloud". In: *ACM SIGPLAN Notices*. Volume 48. 4. ACM. 2013, pages 461–472.

[21] A. Madhavapeddy, R. Mortier, R. Sohan, T. Gazagnaire, S. Hand, T. Deegan, D. McAuley, and J. Crowcroft. "Turning down the LAMP: software specialisation for the cloud". In: *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, HotCloud*. Volume 10. 2010, pages 11–11.

[22] A. Madhavapeddy and D. J. Scott. "Unikernels: Rise of the virtual library operating system". In: *Queue* 11.11 (2013), page 30.

[23] K. Mantripragada, A. Binotto, L. Tizzei, and M. Netto. "A Feasibility Study of Using HPC Cloud Environment for Seismic Exploration". In: *77th EAGE Conference and Exhibition 2015*. 2015.

[24] P. Markthub, A. Nomura, and S. Matsuoka. "Using rCUDA to Reduce GPU Resource-assignment Fragmentation caused by Job Scheduler". In: *15th International Conference on Parallel and Distributed Computing, Applications and Technologies*. 2014. DOI: 10.1109/PDCAT.2014.26.

[25] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. "Innovative Instructions and Software Model for Isolated Execution". In: *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '13. Tel-Aviv, Israel: ACM, 2013, 10:1–10:1.

[26] C. Neuhaus, F. Feinbube, D. Janusz, and A. Polze. "Secure Keyword Search over Data Archives in the Cloud: Performance and Security Aspects of Searchable Encryption". In: *5th International Conference on Cloud Computing and Services Science,* ACM. Lisbon, Portugal, 2015.

[27] C. Neuhaus, F. Feinbube, and A. Polze. "A Platform for Interactive Software Experiments in Massive Open Online Courses". In: *Journal of Integrated Design and Process Science* 18.1 (2014), pages 69–87.

[28] C. Neuhaus, F. Feinbube, A. Polze, and A. Retik. "Scaling Software Experiments to the Thousands". In: *CSEDU 2014 - Proceedings of the 6th International Conference on Computer Supported Education, Volume 1, Barcelona, Spain, 1-3 April, 2014*. 2014, pages 594–601.

[29] C. Neuhaus, M. von Löwis, and A. Polze. "A Dependable and Secure Authorisation Service in the Cloud". In: *CLOSER*. 2012, pages 568–573.

[30] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt. "Rethinking the library OS from the top down". In: *ACM SIGPLAN Notices* 46.3 (2011), pages 291–304.

[31] D. Schatzberg, J. Cadden, O. Krieger, and J. Appavoo. "A way forward: enabling operating system innovation in the cloud". In: *Proceedings of the 6th USENIX conference on Hot Topics in Cloud Computing*. USENIX Association. 2014, page 4.

[32] L. Shi, H. Chen, J. Sun, and K. Li. "vCUDA: GPU-accelerated high-performance computing in virtual machines". In: *Computers, IEEE Transactions on* 61.6 (2012), pages 804–816.

[33] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors". In: *ACM SIGOPS Operating Systems Review*. Volume 41. 3. ACM. 2007, pages 275–287.

[34] K. Tian, Y. Dong, and D. Cowperthwaite. "A full GPU virtualization solution with mediated pass-through". In: *Proc. USENIX ATC*. 2014.

[35] VMware, Inc. *VProbes Programming Reference*. Technical report. 2011.

[36] F. Westphal, S. Axelsson, C. Neuhaus, and A. Polze. "VMI-PL: A monitoring language for virtual platforms using virtual machine introspection". In: *Digital Investigation* 11.S - 2 (2014), pages 85–94.

[37] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose. "Performance evaluation of container-based virtualization for high performance computing environments". In: *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*. IEEE. 2013, pages 233–240.

[38] Xen Project. *The Next Generation Cloud: The Rise of the Unikernel*. Technical report. http://xenproject.org, 2015.

[39] S. Xiao, P. Balaji, Q. Zhu, R. Thakur, S. Coghlan, H. Lin, G. Wen, J. Hong, and W.-c. Feng. "VOCL: An Optimized Environment for Transparent Virtualization of Graphics Processing Units". In: *Proceedings of 1st Innovative Parallel Computing (InPar)*. 2012, pages 1–12.

[40] E. Zhai, G. D. Cummings, and Y. Dong. "Live migration with pass-through device for Linux VM". In: *OLS'08: The 2008 Ottawa Linux Symposium*. 2008, pages 261–268.

# A Branch-and-Bound Approach to Virtual Machine Placement

Dávid Bartók and Zoltán Ádám Mann

Department of Computer Science and Information Theory
Budapest University of Technology and Economics

Finding the best mapping of virtual machines to physical machines in cloud data centers is a very important optimization problem, with huge impact on costs, application performance, and energy consumption. Although several algorithms have been suggested to solve this problem, most of them are either simple heuristics or use off-the-shelf, mostly integer linear programming (ILP) solvers. In this paper, we propose a new approach: a custom branch-and-bound algorithm that exploits problem-specific knowledge in order to improve effectiveness. As shown by empirical results, the new algorithm performs better than state-of-the-art general-purpose ILP solvers.

## 1 Introduction

As cloud data centers (DCs) serve an ever-growing demand for computation, storage, and networking capacity, their operation is becoming a crucial issue. The energy consumption of DCs is of special importance because of both its environmental impact and its contribution to operational costs. According to a recent study, DC electricity consumption in the USA alone will increase to 140 billion kWh per year by 2020, costing US businesses 13 billion USD annually in electricity bills and emitting nearly 100 million tons of $CO_2$ per year [16].

In order to reduce energy consumption, DC operators use a combination of several techniques. Virtualization technology enables the safe co-existence of multiple applications packaged as virtual machines (VMs) on a single physical machine (PM), thus allowing high utilization of physical resources. Live migration makes it possible to move a working VM from one PM to another without noticeable downtime. Since the load of VMs fluctuates over time, this enables DC operators to flexibly react to such changes. In times of low demand, VMs can be consolidated to a low number of PMs, and the remaining PMs can be switched to a low-power state, leading to considerable energy savings. When load starts to rise, some PMs must be switched back to normal mode again so that VMs can be spread across a higher number of PMs.

Finding the best VM placement for the current load level is a tough optimization problem. First of all, multiple resource types must be taken into account, e.g., CPU, memory, disk, and network bandwidth. PMs have given capacity and VMs have given load along these dimensions, and this must be taken into account in VM placement. Moreover, the migration of VMs has a non-negligible overhead in the form of additional network traffic and additional load on the affected PMs. Thus, excessive migrations should be avoided.

In the past couple of years, several different approaches have been proposed for the VM placement problem. From an algorithmic point of view, these can be mostly grouped into two categories: (i) heuristics without any performance guarantees or theoretical underpinning and (ii) exact algorithms using off-the-shelf mathematic programming – mostly integer linear programming (ILP) – solvers [14]. It is dangerous to rely solely on heuristics because in some cases they can lead to extremely high costs or dramatic performance degradation of the involved applications [15]. On the other hand, the exact algorithms suggested so far all suffer from serious scalability issues, limiting their applicability to small problem instances.

In this paper, we propose a new approach, with the aim of finding a good compromise between practical applicability and theoretical soundness. Our approach is based on branch-and-bound, just like typical ILP solvers. However, in contrast to general-purpose ILP solvers, we can make use of problem-specific knowledge to make the search more effective. This is achieved by crafting customized procedures for controlling the branching behavior, custom bounding techniques etc.

## 2 Previous work

Several problem formulations have been suggested for the VM placement problem. They almost always include computational capacity of PMs and computational load of VMs. In fact, in many works, this is the only dimension that is considered [1, 2, 3, 4, 6, 9, 10, 12, 22, 23]. Other authors included, beside the CPU, also some other resources like memory, I/O, storage, or network bandwidth [5, 7, 8, 21, 25].

Different objective or cost functions have been proposed. The number of active PMs is often considered because it largely determines the total energy consumption [3, 4, 6, 8, 23, 25]. Another important factor that some works considered is the cost of migration of VMs [6, 8, 20, 22].

Concerning the used algorithmic techniques, most previous works apply simple heuristics. These include packing algorithms inspired by results on the related bin-packing problem, such as First-Fit, Best-Fit, and similar algorithms [2, 3, 4, 9, 11, 13, 22, 23], other greedy heuristics [17, 24] and straight-forward selection policies [1, 18], as well as meta-heuristics [7, 8].

Some exact algorithms have also been suggested. Most of them use some form of mathematical programming to formulate the problem and then apply an off-the-shelf solver. Examples include integer linear programming [1] and its variants like binary integer programming [5, 13] and mixed integer non-linear programming [9]. Unfortunately, all these methods suffer from a scalability problem, limiting their applicability to small-scale problem instances.

## 3 Problem model

Let $P$ denote the set of available PMs and $V$ the set of VMs hosted in the DC. We consider $d$ dimensions or resource types; e.g., if CPU capacity and memory are considered, then $d = 2$. The *capacity* of each PM and the *load* of each VM is a $d$-dimensional vector. For $p \in P$, its capacity is denoted by $\mathtt{cap}(p) \in \mathbb{R}_+^d$, and for $v \in V$, its load is denoted by $\mathtt{load}(v) \in \mathbb{R}_+^d$. Further, let $|P| = m$ and $|V| = n$.

The DC operator regularly re-optimizes the placement of the VMs in order to adapt to changes [20]. The current placement is given by $\mathtt{map}_0 : V \to P$. Our aim is to determine a new mapping $\mathtt{map} : V \to P$, subject to capacity constraints

$$\forall p \in P : \sum_{v:\mathtt{map}(v)=p} \mathtt{load}(v) \leqslant_d \mathtt{cap}(p), \tag{1}$$

where $\leqslant_d$ is a relation between $d$-dimensional vectors; $(x_1, \ldots, x_d)^T \leqslant_d (y_1, \ldots, y_d)^T$ if and only if for each $1 \leqslant i \leqslant d$, $x_i \leqslant y_i$. $\mathtt{map}_0$ may not satisfy the capacity constraints; even if it satisfied them at the time it was computed, the change in VM loads since then may have rendered it invalid. A PM is *active* if it hosts at least one VM, i.e.,
$p \in P$ is active if $\exists v \in V, \mathtt{map}(v) = p$. The number of active PMs is $\mathtt{act}(\mathtt{map})$. Since energy consumption is largely determined by the number of active PMs, we should minimize $\mathtt{act}(\mathtt{map})$.

A *migration* of $v \in V$ occurs if $\mathtt{map}(v) \neq \mathtt{map}_0(v)$. The number of migrations caused by $\mathtt{map}$ is given by $\mathtt{mig}(\mathtt{map})$. Because of the overhead caused by migrations, we should minimize $\mathtt{mig}(\mathtt{map})$ as well. We combine the two minimization objectives in a single cost function:

$$f(\mathtt{map}) = \alpha \cdot \mathtt{act}(\mathtt{map}) + \mu \cdot \mathtt{mig}(\mathtt{map}), \tag{2}$$

where $\alpha$ and $\mu$ are given non-negative weights defining the relative importance of the two optimization goals. In addition, we require the number of migrations to be below a given limit:

$$\mathtt{mig}(\mathtt{map}) \leqslant K, \tag{3}$$

where $K$ is a given non-negative number. This is sensible because too many migrations make the solution practically infeasible [19]; thus, mappings that would cause too many migrations must be excluded even if they lead to few active PMs and thus to good overall objective value.

To sum up, our aim is to determine a new mapping $\mathtt{map}$ that minimizes (2), subject to constraints (1) and (3).

## 4 Integer programming solution

As a baseline, we formulate the problem as an integer program and solve it with an off-the-shelf ILP solver.

Indexing VMs as $v_i$ ($i = 1, \dots, n$) and PMs as $p_j$ ($j = 1, \dots, m$), the following binary variables are introduced:

$$\mathrm{Alloc}_{i,j} = \begin{cases} 1 & \text{if } v_i \text{ is allocated on } p_j \\ 0 & \text{otherwise} \end{cases}$$

$$\mathrm{Active}_j = \begin{cases} 1 & \text{if } p_j \text{ is active} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathrm{Migr}_i = \begin{cases} 1 & \text{if } v_i \text{ is migrated} \\ 0 & \text{otherwise} \end{cases}$$

Using these variables, the integer program can be formulated as follows ($i = 1, \dots, n$ and $j = 1, \dots, m$):

$$\min \quad \alpha \cdot \sum_{j=1}^{m} \mathrm{Active}_j + \mu \cdot \sum_{i=1}^{n} \mathrm{Migr}_i \tag{4}$$

$$\text{s. t.} \quad \sum_{j=1}^{m} \mathrm{Alloc}_{i,j} = 1 \qquad \forall i \tag{5}$$

$$\mathrm{Alloc}_{i,j} \leqslant \mathrm{Active}_j \qquad \forall i, j \tag{6}$$

$$\sum_{i=1}^{n} \mathrm{load}(v_i) \cdot \mathrm{Alloc}_{i,j} \leqslant_d \mathrm{cap}(p_j) \qquad \forall j \tag{7}$$

$$\mathrm{Migr}_i = 1 - \mathrm{Alloc}_{i,\mathrm{map}_0(v_i)} \qquad \forall i \tag{8}$$

$$\sum_{i=1}^{n} \mathrm{Migr}_i \leqslant K \tag{9}$$

$$\mathrm{Alloc}_{i,j}, \mathrm{Active}_j, \mathrm{Migr}_i \in \{0, 1\} \qquad \forall i, j \tag{10}$$

The objective function (4) is the same as before, consisting of the number of active PMs and the number of migrations. Equation (5) ensures that each VM is allocated to exactly one PM, whereas constraint (6) ensures that for a PM $p_j$ to which at least one VM is allocated, $\mathrm{Active}_j = 1$. Together with the objective function, this ensures that $\mathrm{Active}_j = 1$ holds for *exactly* those PMs that accommodate at least one VM.

Constraint (7) is a straight-forward formulation of constraint (1) in terms of the binary variables $\mathrm{Alloc}_{i,j}$. Equation (8) determines the values of the $\mathrm{Migr}_i$ variables and constraint (9) corresponds to constraint (3).

## 5 Branch-and-bound algorithm

Our algorithm does not use the binary variables introduced for the ILP approach, but operates directly on the $\mathrm{map}$ function. It works with partial solutions, in which $\mathrm{map}(v)$ is defined for a subset of the VMs, and traverses the space of partial solutions in a tree-like manner. For a partial solution, its children in the tree are obtained by selecting a VM that is not mapped yet and trying to map it to all PMs that have sufficient free capacity to host it: for each such PM, a different child partial solution is obtained.

The search starts with all VMs unmapped (the root of the tree), and goes down the tree by mapping one more VM in each step. If all VMs are mapped, then a solution has been found, corresponding to a leaf of the tree. The best solution that has been found so far (*best_so_far*), along with its cost (*best_cost_so_far*), is maintained throughout the algorithm. If the current branch of the search tree cannot be continued or there is no point in doing so, then the algorithm backtracks. This happens in the following cases:

- A leaf has been reached.
- The current partial solution has become infeasible, i.e.,
  - either there is a VM for which no PM has sufficient free capacity,
  - or the number of migrations exceeds the limit.
- All children of the current partial solution have been processed.
- The cost of any solution that extends the current partial solution is surely not lower than the cost of the best solution found so far.

In each of these cases, the algorithm backtracks by undoing the last VM mapping decision, i.e., going back to the parent node in the tree, essentially unallocating the last VM. Afterwards, the next child of the parent is tried, i.e., a new PM is selected for the unallocated VM. When the search would need to backtrack from the root, the algorithm terminates.

The skeleton of the branch-and-bound procedure is shown in Algorithm 1. In the following, the non-trivial parts are described in more detail.

## 5.1 Incremental computations

During the algorithm, many details of the current partial solution are needed, e.g., its cost. Such characteristics can be simply computed directly from the partial solution itself. However, it is much more efficient to compute them incrementally. For example, we maintain the cost of the current partial solution in a variable, and whenever we go up or down in the tree, the necessary change is made to the stored cost value. This way, determining the cost of the current partial solution takes $O(1)$ steps instead of $O(n)$, which is an important difference as this is needed many times.

Beside the cost of the current partial solution, the following characteristics are maintained and incrementally updated:

- The number of migrations.
- The remaining free capacity of each PM.
- For each VM, the set of PMs that still have enough free capacity to host it.

**Algorithm 1:** Branch-and-bound procedure

```
loop
    if all VMs mapped and cost < best_cost_so_far then
        update best_so_far and best_cost_so_far;
    end
    if all VMs mapped or infeasible or all children visited or min_cost ⩾
      best_cost_so_far then
        // backtrack
        if we are in the root then
            return best_so_far
        end
        move back to parent;
    end
    if no VM selected yet then
        select VM;
    end
    move to next child;
end
```

## 5.2 VM selection

VMs can be selected in any order, but this order may have considerable impact on the running time of the algorithm. As the primary criterion for selecting the next VM, we use the first-fail principle, a common approach in constraint satisfaction algorithms: we select the VM with the lowest number of PMs that can host it. This helps to keep the number of children of the nodes of the tree (the branching factor) low and thus the whole tree relatively small.

There can be several VMs with the same number of possible hosting PMs, so we also apply a secondary strategy for tie-breaking: VMs with higher load are preferred. Just like in bin-packing, where sorting the items in decreasing order is known to improve the performance of packing algorithms, here it is also sensible to place the biggest VMs first.

In our case, VM loads are multi-dimensional, so it is not clear what is "bigger." We implemented multiple strategies for sorting d-dimensional vectors:

- Using the lexicographic order of the vectors.
- According to the maximum of the dimensions.
- According to the sum of the dimensions.

## 5.3 PM selection

After having selected a VM $v$, the PMs that can host it must be tried one after the other. Again, the order in which the PMs are tried may impact the performance of the algorithm.

One possibility is to sort the PMs according to their remaining capacity. Again, these are multi-dimensional vectors, so we implemented the same sorting strategies as for VMs, with the single difference that empty PMs are put at the end in order to foster better utilization of PMs that are already on.

Another idea is to start with the PM on which $v$ resides according to $map_0$. Since this strategy only defines the PM that should be tried first, it can also be combined with any of the sorting strategies, which will then determine the order of the remaining PM candidates.

## 5.4 Lower bound on the cost

In Algorithm 1, $min\_cost$ denotes a lower bound on the cost of any solution that can arise as an extension of the current partial solution. If $min\_cost$ is not less than the best cost found so far, then we can backtrack from the current subtree. The question is how to compute a (non-trivial) lower bound.

Let us consider a partial solution, in which a subset $V_1 \subset V$ of the VMs have already been allocated to a subset $P_1 \subset P$ of the PMs. Let $k_1$ denote the number of migrations that been made when allocating the VMs of $V_1$. We have to allocate the remaining VMs of $V_2 := V \setminus V_1$ with at most $K' := K - k_1$ migrations. Ideally, we would like to find the minimal cost according to equation (2), given the constraints (1) and (3) and given the current partial allocation. This is a tough problem. Luckily, we just need a lower bound. This can be achieved by considering a relaxation of the problem: constraint (1) – the capacity constraints – will be disregarded.

The resulting problem is: given the current partial solution, what is the best cost in terms of the objective (2) that can be achieved by the allocation of $V_2$, if at most $K'$ further migrations are allowed? A cost of $\alpha \cdot |P_1| + \mu \cdot k_1$ has already been incurred. For the remaining VMs, since the number of migrations is constrained, this limits how much the new mapping can differ from $map_0$.

Let $C_0$ be the cost of mapping each remaining VM as in $map_0$. If $P_2$ is the set of PMs in $P \setminus P_1$ that are used by $map_0$ for mapping $V_2$, i.e., $P_2 = \{p \in P \setminus P_1 : \exists v \in V_2,\ map_0(v) = p\}$, then $C_0 = \alpha \cdot (|P_1| + |P_2|) + \mu \cdot k_1$. For a mapping with lower costs, some PMs must be emptied, i.e., all their VMs migrated to other PMs. This decreases the cost if the number of VMs that have to be migrated is less than $\alpha/\mu$. In order to empty the maximum number of PMs, PMs with the least number of VMs should be emptied. Therefore, Algorithm 2 delivers optimal result for the relaxed problem.

**Algorithm 2:** Optimal solution for the relaxed problem

```
foreach $p \in P_2$ do
    $a(p) := |\{v \in V_2 : map_0(v) = p\}|$;
end
sort $P_2$ in ascending order of $a(p)$;
$i = 1$;
$mig = 0$;
loop
    let $p$ be the $i$th element of $P_2$;
    if $a(p) \geqslant \alpha/\mu$ or $mig + a(p) > K'$ then
        return
    end
    // empty p
    $mig \mathrel{+}= a(p)$;
    $i$ ++;
    if $i > |P_2|$ then
        return
    end
end
```

This can be simplified with the following ideas: (i) the actual mapping[1] delivered by the algorithm is not interesting, only its cost; (ii) the $a(p)$ values are typically small non-negative integers. For any non-negative integer $j$, let $b_j$ denote the number of PMs in $P_2$ with $a(p) = j$, i.e., $b_j := |\{p \in P_2 : a(p) = j\}|$. Let $J$ denote the highest $j$ for which $b_j > 0$.

**Algorithm 3:** Simplified algorithm for the relaxed problem

```
$cost = C_0$;
$mig = 0$;
$j = 0$;
while $j \leqslant J$ and $j < \alpha/\mu$ and $mig < K'$ do
    $t := \min(b_j, \lfloor (K' - mig)/j \rfloor)$;
    $mig \mathrel{+}= t \cdot j$;
    $cost \mathrel{-}= (\alpha \cdot t - \mu \cdot t \cdot j)$;
    $j$ ++;
end
```

[1]What the algorithm returns is actually not a real mapping because it does not determine where to place the migrated VMs. Since the capacity constraints do not have to be observed now, this does not matter: they could be placed on any of the used PMs.

Algorithm 3 is the simplified version. We just have to iterate through the $(j, b_j)$ numbers. For the first couple of $j$ values, all $b_j$ PMs with $a(p) = j$ can be emptied, followed by a case in which only some $t < b_j$ PMs can be emptied, resulting in $mig = K'$. Using Algorithm 3, the lower bound on the cost can be easily and quickly computed, if the values of $C_0$, $J$, and the $b_j$ numbers are maintained.

## 5.5 Trading off running time and solution quality

All techniques so far help to reduce the running time of the algorithm on typical problem instances, without sacrificing optimality. However, the running time may still be too high for practical applicability. The following techniques reduce the running time further, but without guaranteeing optimality.

**Symmetry breaking**
In a real DC, it is common to have many PMs of the same type. As long as they do not host any VMs, their capacity is the same, introducing some symmetry in the problem. When looking for a host for the current VM $v$, of course the one on which $map_0$ maps $v$ must be handled separately, but all others that have the same capacity are equivalent choices – at least "almost equivalent," as we will see. Hence, it suffices to try just one of them. For example, if there are 100 PMs, all of the same type, then two of them must be tried ($map_0(v)$ and one of the others) instead of 100, yielding a speedup of a factor 50.

Unfortunately, the PMs in question are not fully equivalent because they host different VMs initially (i.e., according to $map_0$). Placing $v$ on one of the PMs may require one of the VMs that was initially on that PM to be migrated to another PM, whereas placing $v$ on another PM may not lead to migrations, for example. By considering only one of these PMs for $v$, the search is not complete anymore: optimality is not guaranteed. Nevertheless, it can be a good heuristic.

**Discarding small improvement possibilities**
If we do not strive for optimality, then a sensible goal is to strive for a solution with cost at most $\gamma$ times the optimum, where $\gamma > 1$ is some given constant. Recall from Algorithm 1 that we backtrack if $min\_cost \geqslant best\_cost\_so\_far$. Now, this condition can be changed to $min\_cost \geqslant best\_cost\_so\_far/\gamma$, resulting in more aggressive pruning. The justification is that either $best\_cost\_so\_far$ is already within $\gamma$ times the optimum, in which case we do not need any further search, or otherwise the condition $min\_cost \geqslant best\_cost\_so\_far/\gamma$ implies that $min\_cost$ is higher than the optimum, so that pruning this part of the search tree does not remove the optimum.

**Limiting the runtime**
The most drastic measure is to simply stop the search after some given time limit, and return the best allocation found so far.

### 5.6 Further remarks

The algorithm can be easily extended to accommodate further constraints in the form of additional pruning rules. E.g., colocation or anti-colocation requirements may exist for certain sets of VMs. These can be ensured by removing the non-compliant options from the list of PM candidates for each VM. E.g., if VMs $v_1$ and $v_2$ must not be colocated and the algorithm decides to map $v_1$ to PM $p$, then $p$ must be removed from the list of possible PMs of $v_2$.

The algorithm works with an arbitrary number of dimensions $d$. Considering the impact on the running time, the steps of the algorithm are either agnostic of the value of $d$, or have a linear runtime in $d$. Further, $d$ is small in practice. Thus, there is no combinatorial explosion with respect to $d$.

## 6 Evaluation

We compare, by means of simulation experiments, three algorithms. The first two methods use off-the-shelf ILP solvers on the ILP formulation of Section 4, as suggested so far in the literature. The solvers are: lp_solve[2] 5.5.2, one of the leading free open-source packages and Gurobi[3] 6.0.5, a successful commercial product. The third method is our branch-and-bound (BB) algorithm.
All measurements were carried out on a desktop PC with 2.6 GHz Pentium E5300 Dual-Core CPU and 3 GB DDR2 800MHz RAM, running MS Windows 7.

Problem instances were generated in the following way. The number of dimensions, $d$, is set to 2. There are 4 PM types; each PM belongs to a randomly selected PM type. In each dimension, the capacity of PM types is randomly generated between 8 and 14, whereas the load of each VM is randomly taken between 1 and 5. The number of PMs and VMs was determined for each experiment separately (see below), varying between 25 and 4000. The number of allowed migrations, $K$, is set to 10 percent of the number of PMs. The weights in the cost function are $\alpha = 10$ and $\mu = 1$.

The initial allocation of VMs to PMs is generated in two steps. First, each VM is randomly mapped on one of the PMs. Such a random mapping may lead to an unrealistically high number of overloaded PMs that cannot be repaired with the limited number of migrations. Hence, in a second step, the First-Fit heuristic is used to pack the VMs into the PMs, with the extension that VMs that did not fit into any PM remain on the PM determined by the random placement. The result is a mapping that has likely few overloaded PMs and some room for consolidation; hence, it models well the typical initial mapping for a VM placement re-optimization algorithm.

[2]http://lpsolve.sourceforge.net/5.5/.
[3]http://www.gurobi.com/.

Each algorithm is run on each problem instance with a timeout of 60 seconds. All presented numbers are the median of 10 measurements. Moreover, we also present lower bounds for the optimum; these were obtained by applying the bounding method of Section 5.4 before any branching has taken place.

## 6.1 Parameter tuning

First, we aimed at finding good settings for the parameters of the BB algorithm. We used randomly generated problem instances with $m$ varying between 25 and 450 and $n$ varying between 50 and 900. Most of the techniques built into the algorithm proved to be indeed useful. The only exception was the technique described in Section 5.5 to cut off branches with small improvement possibilities. The reason why this did not help is probably that – as shown below – the algorithm quickly finds solutions that are quite near the optimum, so that only small improvements are possible afterwards.

**Table 1:** Configuration of the branch-and-bound algorithm

| Technique | Used variant |
|---|---|
| VM selection | First-fail; tie-breaking: sorting according to the maximal dimension |
| PM selection | Initial PM first; rest sorted lexicographically |
| Lower bound | Used as described |
| Symmetry breaking | Used as described |
| Pruning small improvements | Not used ($\gamma = 1$) |

The configuration that turned out to be best and was used in the later experiments is shown in Table 1.

## 6.2 Comparison

Our main objective was to assess the scalability of the three algorithms. To that end, we considered problem instances of increasing size. For this experiment, we fixed the ratio of VMs to PMs to 2, and increased the number of PMs from 25 to 2000, with the number of VMs ranging from 50 to 4000.

To enhance visibility, the results are split into two figures. Figure 1 shows results for problem instances with $n \leqslant 400$, whereas Figure 2 shows the results for bigger problem instances.

In Figure 1, all algorithms perform very similarly for the smallest problem instances. For $n \geqslant 150$, lp_solve fails to deliver a solution. The other two algorithms continue to deliver solutions, with Gurobi performing slightly better than BB for $150 \leqslant n \leqslant 300$. However, BB closes in on Gurobi at around $n = 350 \ldots 400$. In

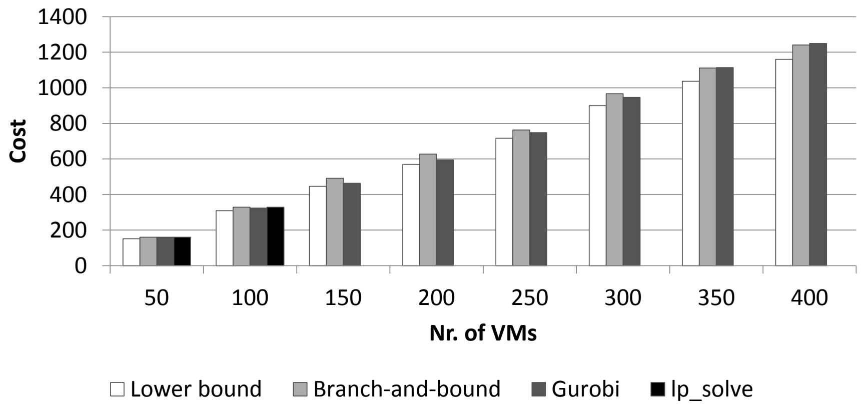

**Figure 1:** Scalability results on small problem instances

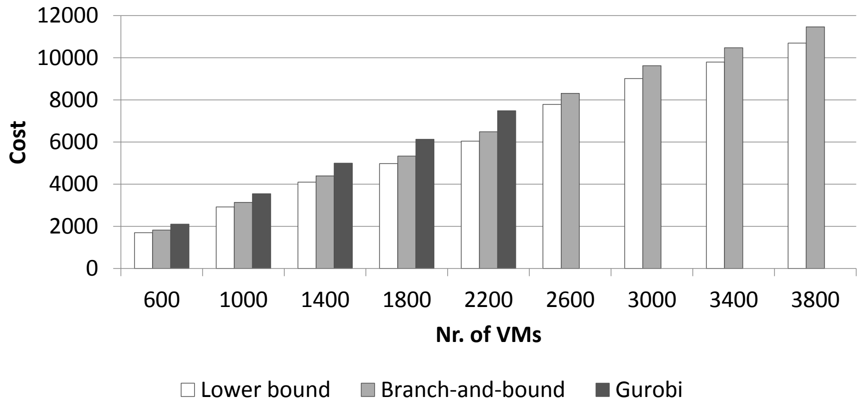

**Figure 2:** Scalability results on big problem instances

Figure 2 we see that for $n \geqslant 600$, BB already consistently outperforms Gurobi, with the latter increasingly drifting away from the optimum. After $n \geqslant 2600$, Gurobi fails to find a valid solution within the given time limit. BB on the other hand continues to deliver results.

The quality of the results found by BB is excellent: they are in most cases within 10 percent of the lower bound, and therefore, also within 10 percent of the optimum.

Finally, we assessed the effect of the load density (the $n/m$ ratio). With 500 PMs, we varied the number of VMs from 500 (lightly loaded DC) to 1500 (highly loaded DC). As can be seen in Figure 3, BB consistently outperforms Gurobi for all densities (lp_solve did not produce valid results in this range).

In our future work, we plan to undertake a more detailed empirical analysis of the algorithm's performance, also comparing it with other algorithms on more realistic test data. Unfortunately, lacking generally accepted benchmarks, this must be done in an ad-hoc manner.

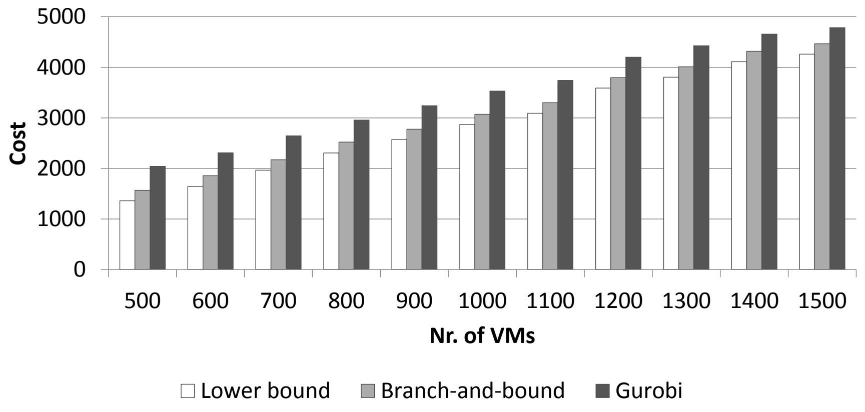

**Figure 3:** Instances with different density ( $m = 500$ is constant)

## Acknowledgment

This work was partially supported by the Hungarian Scientific Research Fund (Grant Nr. OTKA 108947).

## References

[1] D. M. Batista, N. L. S. da Fonseca, and F. K. Miyazawa. "A set of schedulers for grid networks". In: *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC'07)*. 2007, pages 209–213.

[2] A. Beloglazov, J. Abawajy, and R. Buyya. "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing". In: *Future Generation Computer Systems* 28 (2012), pages 755–768.

[3] A. Beloglazov and R. Buyya. "Energy efficient allocation of virtual machines in cloud data centers". In: *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. 2010, pages 577–578.

[4] N. Bobroff, A. Kochut, and K. Beaty. "Dynamic Placement of Virtual Machines for Managing SLA Violations". In: *10th IFIP/IEEE International Symposium on Integrated Network Management*. 2007, pages 119–128.

[5] R. Bossche, K. Vanmechelen, and J. Broeckhove. "Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads". In: *IEEE 3rd International Conference on Cloud Computing*. 2010, pages 228–235.

[6] D. Breitgand and A. Epstein. "SLA-aware placement of multi-virtual machine elastic services in compute clouds". In: *12th IFIP/IEEE International Symposium on Integrated Network Management*. 2011, pages 161–168.

[7] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu. "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing". In: *Journal of Computer and System Sciences* 79 (2013), pages 1230–1242.

[8] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. "Resource pool management: Reactive versus proactive or let's be friends". In: *Computer Networks* 53.17 (2009), pages 2905–2922.

[9] M. Guazzone, C. Anglano, and M. Canonico. "Exploiting VM Migration for the Automated Power and Performance Management of Green Cloud Computing Systems". In: *1st International Workshop on Energy Efficient Data Centers*. Springer, 2012, pages 81–92.

[10] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu. "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures". In: *IEEE 30th International Conference on Distributed Computing Systems*. 2010, pages 62–73.

[11] A. Khosravi, S. K. Garg, and R. Buyya. "Energy and carbon-efficient placement of virtual machines in distributed cloud data centers". In: *Euro-Par 2013*. Springer, 2013, pages 317–328.

[12] D. Lago, E. Madeira, and L. Bittencourt. "Power-aware virtual machine scheduling on clouds using active cooling control and DVFS". In: *Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science*. 2011.

[13] W. Li, J. Tordsson, and E. Elmroth. "Virtual Machine Placement for Predictable and Time-Constrained Peak Loads". In: *Proceedings of the 8th International Conference on Economics of Grids, Clouds, Systems, and Services (GECON 2011)*. Springer, 2011, pages 120–134.

[14] Z. Á. Mann. "Allocation of virtual machines in cloud data centers – a survey of problem models and optimization algorithms". In: *ACM Computing Surveys* 48.1 (2015).

[15] Z. Á. Mann. "Rigorous results on the effectiveness of some heuristics for the consolidation of virtual machines in a cloud data center". In: *Future Generation Computer Systems* 51 (2015), pages 1–6.

[16] Natural Resources Defense Council. *Scaling Up Energy Efficiency Across the Data Center Industry: Evaluating Key Drivers and Barriers*. `http://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf`. 2014.

[17] M. A. Salehi, P. R. Krishna, K. S. Deepak, and R. Buyya. "Preemption-Aware Energy Management in Virtualized Data Centers". In: *5th International Conference on Cloud Computing*. IEEE, 2012, pages 844–851.

[18] L. Shi, J. Furlong, and R. Wang. "Empirical evaluation of vector bin packing algorithms for energy efficient data centers". In: *IEEE Symposium on Computers and Communications*. 2013, pages 9–15.

[19] W. Song, Z. Xiao, Q. Chen, and H. Luo. "Adaptive Resource Provisioning for the Cloud Using Online Bin Packing". In: *IEEE Transactions on Computers* 63.11 (2014), pages 2647–2660.

[20] P. Svärd, W. Li, E. Wadbro, J. Tordsson, and E. Elmroth. *Continuous Datacenter Consolidation*. Technical report. Umea University, 2014.

[21] L. Tomás and J. Tordsson. "An autonomic approach to risk-aware data center overbooking". In: *IEEE Transactions on Cloud Computing* 2.3 (2014), pages 292–305.

[22] A. Verma, P. Ahuja, and A. Neogi. "pMapper: power and migration cost aware application placement in virtualized systems". In: *Middleware 2008*. 2008, pages 243–264.

[23] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari. "Server workload analysis for power minimization using consolidation". In: *Proceedings of the 2009 USENIX Annual Technical Conference*. 2009, pages 355–368.

[24] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. "Sandpiper: Black-box and gray-box resource management for virtual machines". In: *Computer Networks* 53.17 (2009), pages 2923–2938.

[25] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. G., T. Christian, and L. Cherkasova. "1000 islands: an integrated approach to resource management for virtualized data centers". In: *Cluster Computing* 12.1 (2009), pages 45–57.

ISBN 978-3-86956-360-2
ISSN 1613-5652