Autonomic Management of Service Level Agreements in Cloud Computing

By

STEFAN EDWIN KARL FREY

School of Computing, Electronics and Mathematics
PLYMOUTH UNIVERSITY

A thesis submitted to Plymouth University in partial fulfilment for the degree of DOCTOR OF PHILOSOPHY

JANUARY 2019

COPYRIGHT STATEMENT

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the author's prior consent.

I

<div align="right">**ABSTRACT**</div>

Autonomic Management of Service Level Agreements in Cloud Computing

Cloud computing has been one of the major trending topics of recent years in the Information Technology (IT) industry. In contrary to the previous IT service delivery models, where all the services and resources where hosted locally, the idea behind cloud computing is to deliver computing resources or services on-demand over a network on an easy pay-per-use business model. This paradigm change included almost all areas of the modern IT landscape, be it the storage and sharing of data for example with services like Dropbox, Microsoft OneDrive or Google Drive or creating complete computational environments with Amazon Web Services or the Microsoft Azure Cloud Platform. Due to the lower upfront costs, rapid provisioning, elasticity and scalability, the adoption of cloud services is steadily increasing. Thereby cloud computing nowadays for many companies has become a practical alternative to locally hosted resources and IT services. However, in the currently offered state of cloud computing, there are significant shortcomings in regard to guarantees and the Quality of Service (QoS). In order to make cloud services effectively usable and reliable for enterprises, Service Level Agreements (SLA) are needed, which state the precise level of performance, as well as the manner and the scope of the service provided. This practice, which is widespread in the area of IT services, but is currently of limited use for cloud computing, due to the fact that existing cloud environments offer only rudimentary support and handling of SLAs, if any. Moreover, the classic SLA management approach is a rather static method, whereas due to the dynamic character of the cloud, the QoS attributes respectively service levels must be monitored and managed continuously. In addition, performance indicators and measurement methods for service level objectives in cloud computing have been studied inadequately. The proposed research focuses on cloud specific SLA management as well as the associated functional and non- functional QoS parameters and measurement methods. To address

the shortcomings of the current state of the art cloud computing SLA landscape and enable cloud users to dynamically create, adjust and monitor SLAs for their cloud services, an architecture for providing autonomous management of cloud services together with an easy to use SLA control interface is introduced, to monitor the agreed service performance, facilitate SLA compliance and enable dynamic customer- specific adaptation of SLAs for cloud services. Together with the specially tailored machine readable Adaptable Service Level Objective Agreement language (A-SLO-A), and highly the specialized scaling mechanisms, as well as a SLA scheduling module, the proposed research builds a holistic approach of enabling QoS and SLAs in Cloud Computing.

DEDICATION AND ACKNOWLEDGEMENTS

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and at no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award.

This study was financed with the aid of a research grant from the Federal Ministry of Education and Research (BMBF) of Germany.

Relevant seminars and conferences were regularly attended at which work was presented and several papers prepared for publication, details of which are listed in the appendices.

Word count of this PhD thesis:

Signed .... Date ....

Table of Contents

1

The following chapter starts with the motivation, why the autonomic management of service level agreements in cloud computing is a demanded capability (Section 1.1). Afterwards the limitations and shortcomings of the current cloud computing landscape, as well as the aim and objectives of this thesis are summarized (Section 1.2). Finally the structure is outlined (Section 1.3) and the research grant under which this study has been conducted is stated (Section 1.4).

1.1 Motivation

Cloud computing has been one of the major trending topics of recent years in the Information Technology (IT) industry, and at the latest by the occurrences of the "iCloud Leak" incident in late 2014[28], the term cloud will have arrived in the mind of the general public. In contrary to the older IT service delivery models, where all the services and resources where hosted locally, the idea behind cloud computing is to deliver computing resources or services on-demand over a network on an easy pay-per-use business model[109]. This paradigm change included almost all areas of the modern IT landscape, be it the storage and sharing of data for example with services like Dropbox[76], Microsoft OneDrive[103] or Google Drive[40], or complete computational environments like Amazon Web Services[123] or the Microsoft Azure Cloud Platform[110]. Nearly anything today is capable of being transferred to the cloud. As a result, the investment costs for IT infrastructures or services can be lowered, since it is now obtained from a cloud provider and not bought or provided by the user himself. At the same time modern cloud providers posses an almost inexhaustible amount of computing resources pooled for their customers, so that nearly all requested amounts of resources can be provided within minutes. Due to the lower upfront costs, rapid provisioning, elasticity and scalability, the adoption of cloud services is steadily increasing [58]. Thereby cloud computing nowadays for many companies has become a practical alternative

to locally hosted resources and IT services. According to the analyst view of the Crisp Research AG[6], the overall cloud spendings of Germany will rise from an estimated 10,9 billion Euro in 2015 to approximately 28,4 billion in the next three years. This can be interpreted as a clear trend towards the cloud business.

However, in the current offered state of cloud computing, there are significant shortcomings in regard to guarantees and the quality of offered services. But such guarantees for bough services are desperately needed, especially by enterprises, to make could computing effectively usable[38] and reliable[64]. For this purpose so-called Service Level Agreements (SLA) are needed, which state the precise level of performance, as well as the manner and the scope of the service provided. This practice, which is widespread in the area of IT services, is currently of limited use for cloud computing, due to the fact that existing cloud environments offer only rudimentary support and handling of SLAs, if any. And therefore most providers do not offering any kind of SLA or just generic versions of standardized guarantees, such as availability or service & and helpdesk. For example Amazon Web Services[123], as one of the biggest players in cloud services only gives an availabitliy guarantees for their Elastic Compute Cloud (EC2)[12], which is simply stated as if the achieved monthly availability is below 99.95% the the customer will be given back 10% of the monthly fee in service credits. Additionally up to 30% of the monthly fee will be credited if the falls below availability is below 99.0%. There are no further performance or quality guarantees given so far. This means that without even having to pay credits back the EC2 cloud service can be down for 21.56 minutes every month plus maintenance. This for of SLAs is widespread in the current cloud computing landscape[37] [18] and does not offer any sufficient protection for the customer. In addition, practically unusable services, due to poor performance are not even taken into account. Apart from this is the compensation through service credits is in not proportion to the expected actual financial damage that a company can suffer due to poor availability, which can render this whole SLA meaningless[77]. Cloud users should be given the opportunity to configure related services according to their needs and to obtain customized guarantees. So that users in need can protect particularly important services. This means that cloud providers firstly must be able to provide the functionality of custom SLAs and be able to guarantee the corresponding quality of service parameters and demand a compensation. The following section states the aimes and objectives of this thesis, by analysis of the current state of the research and cloud computing environment in order to enable service level management in cloud computing.

1.2 Aim & Objectives

Literature review on related work in the area of cloud computing guarantees, quality of service and Service Level Agreements shows, that the following limitations exist:

The classic SLA management approach is a rather static method, whereas due to the dynamic character of the cloud, the QoS attributes respectively service levels must be monitored and

managed continuously[96].

Performance indicators[49] and measurement methods for service level objectives in cloud computing have been studied inadequately[66].

The purpose of the presented research has been to investigate the integration of SLAs into cloud computing environments to promote their reliability, transparency, trust and mitigate business concerns. Therefore cloud specific performance indicators, in dependence of cloud attributes such as, on-demand availability, elasticity of cloud resources and deployability were identified. Following, an investigation of traditional SLA managment and its integration into the cloud shows how there is need for changes. To solve the identified problems an autonomic management of SLAs for cloud environments has been proposed.

1.3 Outline

The remainder of this manuscript is structured as follows.

Chapter 2 gives some background information of this study, starting with cloud computing its history, definitions and a reference architecture to build a common knowledge base. Furthermore the foundations of Service Level Agreements, the proces of SLA management as well as the guidelines on the content and preconditions will be given and the SLA lifcycle will be introduced. Additionally the autonomic computing paradigm is introduced and discussed.

Chapter 3 follows up on Service Level Agreements, by illustrating the current cloud computing SLA landscape and introducing cloud specific key performance indicators as basis for cloud service SLAs. Furthermore will the monitoring of cloud quality of service parameters and the controls and management will be elaborated.

Chapter 4 describes the presented approach for autonomic SLA management in cloud computing, starting wiht the introduction of the architecture and its modules. Subsequently the workflow from SLA creation to SLA monitoring and execution is illustrated to demonstrate the application of the autonomic SLA management.

Chapter 5 - Implementation and Evaluations describes the developed prototype and its components. The Chapter follows the structure of the main research phases and presents the three consecutive development stages of the prototype:

- Stage 1 - "ASLAMaaS Front-end" - The graphical user interface to build Cloud SLAs and exported in the machine readable A-SLO-A language.

- Stage 2 - KPI respectively SLA monitoring and enforcement with various management and prediction techniques.

3

- Stage 3 - Cloud Simulator integration for evaluation and proof of concept.

- Stage 4 - Holistic SLA management based on provider prioritisation.

Each section ends with a demonstration of its specific part of the development stage and a evaluation based on three scenarios is given in the end .

Chapter 6 presents the related work, which is divided into 5 categories: Infrastructure Measurements and Cloud Monitoring, SLA Description Languages, Performance Prediction, SLA Enforcement, and Scheduling Mechanisms. The related works of each section is analysed and distinctions to the contributions in this thesis a drawn.

Chapter 7 summarized the presented work and shows the limitations of the proposed solutions. Afterwards the lessons learned during this study and possible future work arising from the research contributions of this thesis is described.

The Appendix shows the contributed research activities accompanying thesis with the published papers and reports.

Nomenclature Within this thesis technical and legal terms are introduced at their first appearance. Additionally a collection of the most important ones are listed in Section XXXXXXXXXX The terms cloud user, cloud consumer and cloud customer are used synonymously throughout this work.

1.4   Research Grant

The research topic "Autonomic SLA Management as a Service (ASLAMaaS)" presented in this report, is supported by a research grant from the German Federal Ministry of Education and Research (BMBF) within the FHprofUnt2012 program. The grant is issued for the period of three years, starting October 2012 and is referenced under the grant number 03FH046PX2.[15]

CLOUD COMPUTING & SERVICE LEVEL AGREEMENTS

This chapter considers the fundamental concepts and terms, which are being utilized in the following chapters. The aim is to give brief explanations and definitions that are necessary for the further comprehension of the topic. Section 2.1 introduces the concept, history and the current state of cloud computing and presents its different cloud models. Section 2.2 discusses the foundations of Service Level Agreements in the IT Industry, as well as to introduce structural and content based pre-conditions and the SLA lifecycle. Section 2.3 explicates the autonomic paradigm and its applications. Finally, Section 2.4 summarizes the basic concepts and scientific classifications of self-adaptive (or autonomic) software systems.

Section 2.1 introduces the concept of component-based software engineering and presents common component models. Section 2.2 discusses the performance of a software system, particularly modeling and measurement of performance characteristics. Section 2.3 explicates alternative instrumentation techniques to inject monitoring probes into a software system. Finally, Section 2.4 summarizes the basic concepts and scientific classifications of self-adaptive (or autonomic) software systems.

2.1 Cloud Computing

After an initial hype cloud computing is establishing itself as an adequate means of providing resources or services on demand. Never before in the history of computers has it been so easy for users to acquire enormous quantities of computational power and resources and put them to use almost immediately. Through this novel way of computing users do not have to manage and maintain the IT assets they use and are not longer bound to the limited local resources they own. In comparison to the conventional approach of using of IT resources, a user in cloud computing gets charged by the provider based upon the extent and time he utilizes or keeps the

resource or service reserved. Through this Äúpay-per-use model, and the rapid provisioning of cloud computing resources new opportunities in terms of dynamization, reduction of costs and investments emerge. The remainder of this chapter will point out the history of cloud computing and will indicate why this presents a relevant research topic in the information technology. The introduction of a reference architecture for cloud computing and the different service models will create the technical background for this report. Finally a discussion of the advantages and disadvantages shall complete the state of the technology as it is.

2.1.1 History of Cloud Computing

Cloud computing is one of the most used buzzwords in information technology and it has become an extremely popular label, for all kinds of Internet and IT services. A simple Google search reveals its magnitude with over a 100 million search results. Cloud computing thereby often is propagated as a new computing paradigm or as a information revolution [26]. The use of the term "cloud computing" really became popular in 2006 when big companies like Google and Amazon (Elastic Compute Cloud) [41] started using it. But the origins can be traced even further back in time into the late 1990s, where MIT Technology Review states the first documented use of the term dates back to 1996 on a Compaq business plan [115]. Also academic papers, which likewise used the term, were published in 1997 [63] [29].

Aside from the terminology, the basic concepts of cloud computing date back even further. In the 1950s large-scale mainframe computers were installed in large companies, schools and government organizations. Because of the colossal hardware infrastructures and their immense costs of operating and maintaining, it was not feasible to grant each user sole access to his own mainframe. Therefore multiple users would have to share access to a mainframes computational power and storage via "dumb terminals". This is directly reflected in the cloud characteristics of shared resources and multi-tenancy. [35]

Cloud Computing is largely based upon virtualization technologies and the Internet. Virtualization enables computer systems to share resources so that one physical resource can act as many virtual instances [74]. This is done by masking the real resources and granting access to them through a abstraction layer. IBM implemented the first virtual machines in the 70s with their operating system called VM. It allowed the division of huge mainframe computers into multiple distinct computers running in the same processing environment. At the same time developments in the telecommunication played a crucial role for the cloud development. The whole concept of utilizing shared resources firstly became reasonably usable for everyone with progression towards fast broadband connections.

Mather et al. [135] describe the emergence of cloud computing as a logical evolution of computing itself and view it as a development of the internet service providers (ISP). Figure 21 shows the evolution towards cloud computing. At first ISPs merely provided Internet access to their customers (ISP 1.0). After the access to the Internet became common, ISPs extended
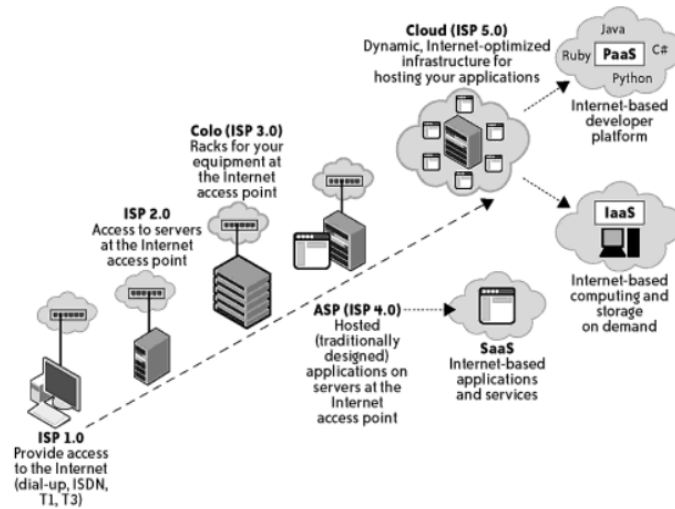
Figure 21: Evolution of Cloud Computing from [135, p.4]

their offerings towards providing additional services like email and access to the servers at their facilities (ISP 2.0). This led to so-called collocation facilities (ISP 3.0), data centers where the ISP provided the infrastructure for the customer servers to be hosted. The next step in the evolution was the application service provider (ASPs), which added additional higher services like customized application for organizations (ISP 4.0). The service delivery model of ASPs may appear very similar to cloud computing, especially to the Software as a Service (SaaS) model, but there is a key difference in the underlying infrastructure. Within the ASP model every customer had his own dedicated infrastructure, so therefore every customer even had his own dedicated server and sole access to it. While cloud computing (ISP 5.0) offers access to shared resources.

At present, according to the Gartner Research Group [61], cloud computing is moving towards productivity and establishing itself as new means of getting IT resources and computational power. Figure 23 shows the Technology Hype Cycle for 2014, where it can be seen that after the initial peak interest and the following disillusionment cloud computing is expected to reach maturity and the plateau of productivity within the next 2 to 5 years. This can be seen as a strong indicator of the increasing importance of cloud computing. In Germany, according to the german Association for Information Technology, Telecommunications and New Media (BITKOM) [5], in 2013 40% of all german companies were using cloud services. Especially large companies with over 2.000 employees increasingly relied on this technology. Overall cloud computing is becoming immensely popular. Juniper Research [116] estimated the overall global cloud service users in 2013 to be 2.4 billion. Their forecast even predicts 3.6 billion cloud users for 2018. Cloud computing is therefore a focal point for researchers in computer science.

7
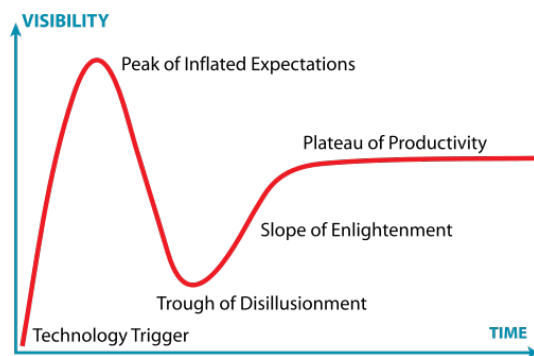
Figure 22: Hype Cycle Phases
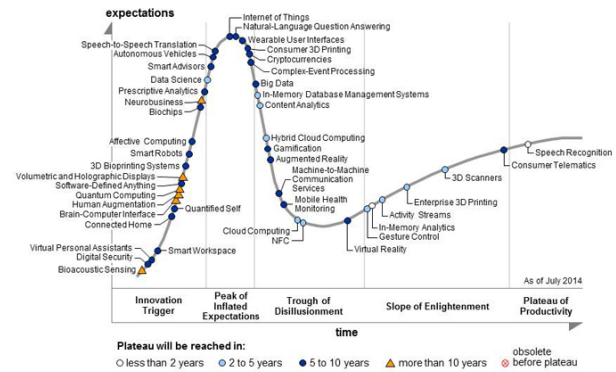
Figure 23: Gartner Group Technology Hype Cycle 2014 [61]

2.1.2 Cloud Computing Definition

Even though cloud computing has become very popular in recent years, it is still difficult to get a generally accepted definition of it. What come closest to that is the definition of the US National Institute of Standards and Technology (NIST), which presents itself as a de facto standard for cloud computing:

"Cloud Computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models and four deployment models" NIST Definition of Cloud Computing [109]

A graphical representation of the cloud model based on the definition from NIST is shown in Figure 24. Here all the essential characteristics, service models and deployment models can be seen at once.

Characteristics

NIST‚Äôs cloud computing definition identifies the following five essential cloud characteristics:

On-demand self service: Cloud customer can provision and manage cloud resources like computing power and network storage without requiring human interaction with a service provider.

Broad network access: Provided resources are accessed via networks (mostly the internet) using standardized protocols.

Resource pooling: The computing and storage resources of a provider are shared between multiple customers (multi-tenant model). A customer doesn't know the exact physical location of the resources he is using, but may specify a location on an higher abstraction level.
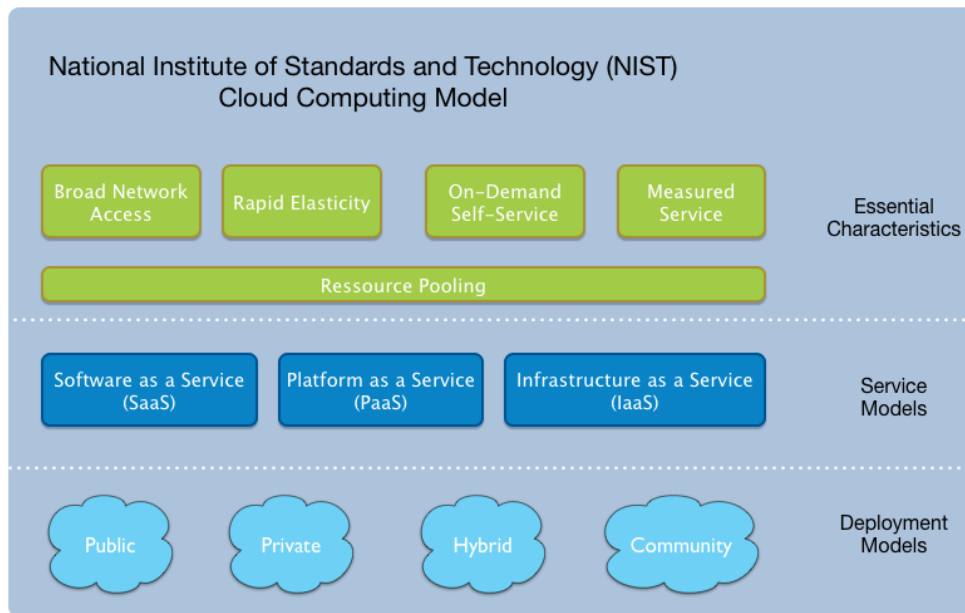
Figure 24: NIST Cloud Computing Model cf. [109]

Rapid elasticity: Resources can be provided elastically and scaled rapidly to fulfill the customers current demand. This can also be done automatically.

*Measured service:*Consumption of resources by the customer gets metered. The cloud customer only pays for the resources and services he actually used. For transparency usage is controlled, monitored and reported to both provider and customer.

2.1.3 Cloud Models

As seen in Figure 24 another distinction is made between the three service models Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Figure 25 below shows the differences between the service models regarding flexibility and provider management.

It becomes evident that the models with increased flexibility in terms of usage come with less provider management and involvement, which means that the customer has to manage them by himself. This gets particularly clear with the Infrastructure as a Service model.

Infrastructure as a Service (IaaS)

Within the IaaS service model typical infrastructure components like virtual machines, CPU power, memory, storage and networking is provided. The user is able to run and deploy software on this resources at will. This may include operating systems. For this the physical infrastructure of the provider is virtualized by technologies like KVM, Xen or VMware and orchestrated into the virtual infrastructure for the user. The user does not control or manage the physical cloud infrastructure but can control and manage the virtual instance together with the operating sys-

9
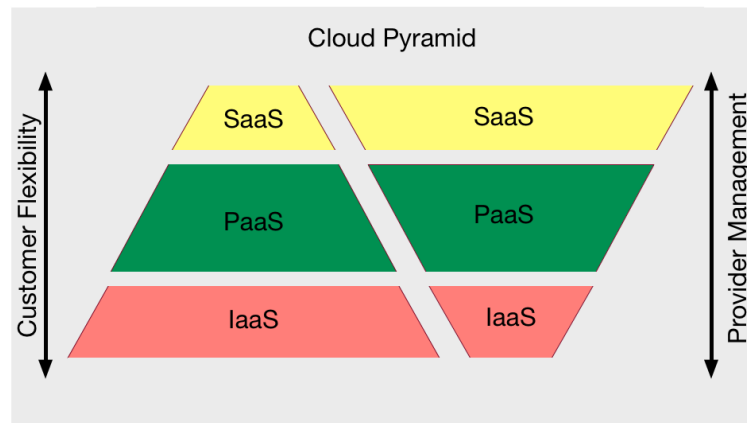
Figure 25: Cloud Pyramid of Flexibility and Management

tem and the components that form the virtual instance, for example host-firewalls, load-balancer and so on. The main difference to traditional hosting servicer like server homing, is that the user only is provided with a virtual instance instead of the physical hardware. For cloud providers this means that they can better divide their resources between customers and so boost overall utilization. The provider charges the customer based on the amount and time of resources he has used on a on-demand model. The provider is responsible for the availability and usability of the infrastructure. Everything else like the reasonable usage, configuration and installation, as well as integration into the customers IT landscape and connection to other system remains with the customer. Often providers offer automatic installation services for operating systems based on virtual machine images. This may require based on the software additional licensing cost but mostly is offered as a free service. Storage as a Service is a IaaS model where only access to disk space is granted. This model become increasingly popular in the last year, since it also enables users to easily share their data with third parties. Best known for this are providers like Dropbox [76], Microsofts OneDrive [103] or Google Drive [40]. Most popular IaaS providers at the moment are Amazon Web Services [123] and Rackspace [112]. Lately Google Compute Engine [32] and Microsoft Windows Azure [110] started their IaaS services.

Platform as a Service (PaaS)

This model aims at developers by providing comprehensive development and run environments to easily create, test and deploy applications. The user gets presented with a pre-configured environment by the provider, which means he does not have to deal with management or install of the virtualized infrastructure. The provider defines and configures as well all the development toolkits and environments, programming languages, APIs. libraries and databases. In extensive PaaS offerings users can immediately start to develop or deploy applications, without even installing tools on their machines. This model therefore enables developers a rapid propagation

of their applications, location independent multi-tenant development and minimal entry effort and cost. Well known PaaS offers are Google App Engine [45] and Microsoft Windows Azure [110].

Software as a Service (SaaS)

In the SaaS model the user is provided with the capability to use the applications running on the providers cloud infrastructure. The users does neither manage, nor control the infrastructure or its components, the operations system or any application capabilities with the exception of settings within the applications. The user usually accesses the rented applications through either the web browser or an adapted program interface. In contrast to traditional software usage the customer subscribes to the service or uses a pay-per-use model. It is quite common that SaaS provider use PaaS or IaaS infrastructures or third party providers to benefit form the high elasticity. Popular SaaS providers are Google Apps [13], Google Docs [39], Microsoft Office 365 [7] and Salesforce.com [118].

Deployment Models

The NIST definition tells apart four different deployment models. As seen in Figure 24 these are: Public cloud, Private cloud, Hybrid cloud and Community cloud.

Private clouds are used exclusively by one single organization, for example by a company for serving its internal needs. These cloud are also often referred to as internal clouds, since the infrastructure is commonly run inside the private network of the using entity. The cloud system may be operated and managed by the organization itself, a third party or a combination of both.

Public clouds represent the exact opposite. Here the usage of the cloud is not limited to one user respectively one organization, instead the services are offered to a larger group of customers or the general public. This cloud type is also called external clouds, since they are managed, hosted and operated by a third-party vendor outside the influence of the customer. The services are commonly accessible through web-applications, web-services or communications protocols like the Remote Desktop Protocol (RDP) or Secure Shell (SSH). A well known provider for public cloud services is the Amazon Elastic Compute Cloud (EC2).

Community clouds are used exclusively by a specific community of consumers. This usually are organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations cf. [109]). The infrastructure my exist on or off premises, as well as may be owned, operated and managed by one or more of the community entities or a third party or any combination of the mentioned before.

Hybrid clouds present a composition of two or more of the mentioned above cloud deployment models. The infrastructures are bound together but still remain unique entities. This is

done by enabling data and application portability through the use of standardized or proprietary technology.

2.1.4   Reference Architecture

Different cloud architectures include various components. In terms of a wide acceptance the NIST proposed architecture is chosen to give an overview of possible components. The NIST architecture identifies five major actors. Table 21 shows these actors and gives the definition of them. The descriptions are taken directly from the NIST Cloud Computing Reference Architecture [92].

Actor	Definition
Cloud Consumer	A person or organization that maintains a business relationship with, and uses service from, Cloud Providers.
Cloud Provider	A person, organization, or entity responsible for making a service available to interested parties.
Cloud Auditor	A party that can conduct independent assessment of cloud services, information system operations, performance and security of the cloud implementation.
Cloud Broker	An entity that manages the use, performance and delivery of cloud services, and negotiates relationships between Cloud Providers and Cloud Consumers.
Cloud Carrier	An intermediary that provides connectivity and transport of cloud services from Cloud Providers to Cloud Consumers.

Table 21: Cloud Computing Actors from NIST [92]

The Cloud Provider is the entity that delivers the cloud service to the Cloud Consumer. Since the delivered service are to be consumed over a broad network access a third party. the Cloud Carrier is used to provide connectivity between the both. This can for example be the Internet Service Provider or in case of a private cloud it would be the IT department responsible for the network. In some cases different roles can be adopted by one organization this may be the case especially with private clouds but also if the Cloud Provider at the same time is a ISP and therefore takes the role of the Cloud Carrier.

A Cloud Provider usually owns one ore more different data centers (facilities). Here the physical hardware for the cloud infrastructures and the environment needed (cooling, electric power, ...) to support it is hosted. In the presented architecture this is grouped together as the *physical resource layer*. Figure 26 shows the overview of the components of the NIST cloud architecture.

In between the *physical resource layer* and the *service layer*, which provides the three different service model (IaaS, PaaS, SaaS) there is the *resource abstraction and control layer*. These three layers form the *Service Orchestration* module which is used to compose and deliver the actual

cloud services. The *service layer* on top is where the Cloud Provider defines the access interfaces for the Cloud Consumer. In the course of this it is possible but not necessary that SaaS applications can be built on top of PaaS components and these again can be built on top of IaaS components. So for example, a PaaS environment could be hosted on top of a virtual machine from an IaaS cloud.

The *resource abstraction and control layer* contains all system and software components that the Cloud Providers uses to abstract the physical resources into virtualized cloud resources. This may include hypervisors or virtual machine monitors, virtual storage, virtual machines and so on. The virtualization in this layer allows resource pooling, dynamic allocation and measurement of the consumed services. Here the control (allocation, access control and usage monitoring) of the cloud resources is realized.
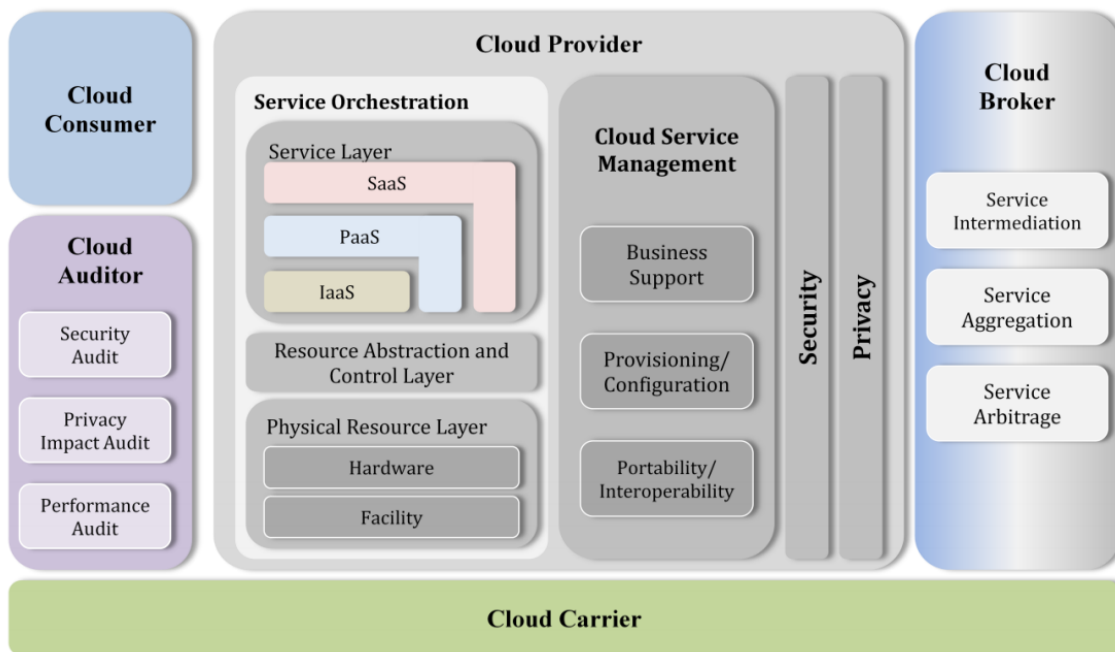
Figure 26: Cloud Computing Reference Architecture from NIST [92]

In addition to this core component, there is the *cloud service management* and modules for *security* and *privacy*. The cloud service management module includes the functions which are needed by the Cloud Provider for the management and operation of the provided cloud services. These are grouped into three different Cloud Provider perspectives. Firstly the *business support*, which includes the processes supporting the service offering and dealing with clients, like the customer management, contact management, accounting & billing and reporting & auditing. Then there are the processes involved in the *provisioning and configuration* of the cloud services. These include the monitoring and metering as well as the SLA management. Lastly there are the mechanisms to support *portability and interoperability*, which shall enable the service customer to

freely merge and compose cloud services between different cloud providers. A detailed description of all these three modules and their individual components can be found in the NIST publication [92, p. 15].

Related Technologies to Cloud Computing

As described earlier cloud computing has evolved from technologies that date back early in the history of computer science.Therefore it is related to different technologies by sharing similar aspects. Following the shared aspects with virtualization, grid computing, utility computing and autonomic computing will be shown.

Virtualization presents itself as one of the core technologies of cloud computing. Virtualization abstracts layers of physical resources from one another, thus making it possible to compose virtual systems out of virtual resources. Trough aggregation of large computational resources across multiple physical machines, virtual resources can be pooled by the provider. Therefore virtualization forms the foundation of cloud computing, as it provides the capability of assigning and reassigning virtual resources dynamically to cloud customers on-demand [147].

Grid Computing aims at creating supercomputer like computing resources by coordinating and utilizing distributed network connected resources. Grid computing initially was driven by scientific applications like protein folding [**?**] or particle simulation [**?**] or the search for pulsars [**?**]. Grid computing always serve a specific task and all participating resources aim to achieve this common objective. Cloud computing appears very similar and shares a lot of the attributes from grid computing, for example the use of distributed resources and abstracted utilization. But it differs in term of dynamic resource provisioning, the abstractions, the business model and applications [52].

Utility Computing contains the business model of "pay-per-use" for computing services. Much like for example electricity the provided computing services are metered and the customer is billed based on his usage. Cloud Computing adopts this billing model and therefore can be seen as a realization of utility computing.

Autonomic Computing was coined by IBM in the early 2000s, where autonomic refers to self-managing computer systems. The aim was to create computing systems that were able to react and respond to internal and external observations without any human interaction. Cloud computing shares some of the aspects considering self-management, like autonomic provisioning of resources with autonomic computing. But instead of aiming for fully autonomous systems with for example self-correction the aim in cloud computing is to reduce management involvement and improve the on demand character.

2.1.5 Cloud Computing Advantages and Disadvantages

Cloud computing comes with many benefits especially for small and medium sized business users. In this section the advantages are described both form perspective of the provider and

consumer. As one of the major benefits of cloud computing the increased agility and adaptability can be seen. Here customers can take advantage of extending their IT capabilities based on the demand to fit the changing requirements. In addition the customer does not have to manage or maintain the underlying infrastructure, which reduces his personnel training efforts and costs and therefore can stronger focus on his core strengths. Cloud computing also makes it possible to obtain enormous computing resources within very short times without having to bear the cost of acquisition. Due to this low up-front investments cloud computing offers a low entry barrier for young companies and simultaneously enabling rapid testing and development. For cloud providers using the virtualization technologies enables them to better utilize their infrastructure, by dividing the distributed physical hardware into virtual resources, since not all of the services in usually use the full provided resources. This reduces the overall Total Cost of Ownership (TCO). Because providing cloud services is the core competence of the cloud provider the offered services usually provide better performance and availability. Especially for SMEs it would be very expensive to reach the same high availability levels of modern cloud computing providers and depending o the usage cloud computing reduces the operating and maintenance cost of the customers services [131]. This is due to the on-demand character of the cloud where the customer can reduce the rented resources of his services with low usage to a minimum. This reduces costs for the customer, while he still is being able to quickly adapt them back in case of increasing demand so that business risks of under-performance is reduced. In addition the further risks like recovery due to outage or hardware failure is also moved to the cloud provider.

Besides the aforementioned benefits cloud computing also bears some disadvantages and risks. The main concern of many cloud users still is security [78]. The primary concern for many users is the loss of control over data or the infrastructure ([19], [149]). This is due to lack of a proper, cloud provider independent security model. In addition the unknown physical location of the rented resources (especially in Public clouds) and the absence of cloud security standards and improper backups bare a risk of data loss. Besides that the use of cloud computing does not always mean a reduction in operational costs. For permanent use classic hosting offerings are significantly cheaper. For example a streaming server based on a m3.xlarge Amazon EC2 instance with 4 vCPU cores (One cCPU provides the equivalent capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor), 15GB memory, 80GB SSD, which is used 24/7 for a whole month. With 500GB of storage and 1TB incoming and 5 TB outgoing network traffic, located in Ireland would costs around 864.06 USD [11]. In contrast a conventional dedicated hosted server with a 4x Intel Core i7 2,66 GHz CPU, 14 GB of memory, a 80GB SSD and 2TB storage, with a 1 Gbit/s connection and unlimited traffic, which amounts to roughly 320TB combined traffic per month, would cost about 505,47 USD [34]. Additionally all cloud services are heavily network dependent, respectively Internet dependent. So the availability and performance of the connection is a core requirement for using cloud services. In case of missing or bad performing connections the provider is not necessarily responsible, since he only has to provide his services

towards his edge of the network, so users have to ensue reliable connections. A additional risk may be the dependency on the vendor and possible vendor lock-in. This occurs due to the lack of interoperability between cloud providers. Virtualization adds abstraction between the users and the physical hardware, but since not all virtualization technologies are inter operable moving from one cloud provider to another may be difficult.

2.2 Service Level Agreements

A service level agreement (SLA) in general is a formal bilateral legal contract between a provider of services and the consumer of such services, that defines the scope and quality expected that the provider commits to deliver to the customer. The Information Technology Infrastructure Library defines SLA as the following:

Service Level Agreement- A formal, negotiated document that defines (or attempts to define) in quantitative (and perhaps qualitative) terms the service being offered to a Customer. Confusion must be avoided over whether the quantitative definitions constitute thresholds for an acceptable service, targets to which the supplier should aspire or expectations that the supplier would strive to exceed. Any metrics included in a Service Level Agreement (SLA) should be capable of being measured on a regular basis and the SLA should record by whom. ITIL V2 [80]

The aim of such contracts is create a reliable, legally binding arrangement from witch both the customer and the provider can benefit. The content of a SLA will typically will cover general service related information like serivce hours and availability, but also will include the performance expectations and desired service outcomes in term of functionality and responsiveness. This also may include incurring costs, information about security and additional terminology. For customers it offers the possibility to guarantee their expected service outcome and in case of noncompliance enable financial compensation. For providers it states clearly all the agreed upon deliverables, as well as the method of delivery, occurring costs, verification and sanctions he has to comply to. So that he can deliver his services based on that and charge the customer without any discrepancies. The other side of the coin is the provider needs to establish SLA management so that he can satisfy the contracts.

Since Service Level Agreements specify the promised respectively the expected performance characteristics of a service, the most important part is the exact description of the service quality (service level). But the creation of Service Level Agreements provides requirements to customers and providers. Customers need to be able to meet certain requirements in order to successfully define SLAs, which are listed briefly below [120].

A customer must:

- Understand the roles and responsibilities that are regulated by the SLA.

- Be able to describe precisely and specific the service to be controlled by the SLA.

- Know the requirements of the controlled services, and define the matching key figures.

- Specify service levels based on the critical performance characteristics of the service.

- Understand the process and procedures of regulated service.

These requirements are necessary so that the customer is able to put in the correct SLAs values, and to understand implications of his decisions. Furthermore, a SLA should fulfill the following tasks:

- Describe the services accurately.

- Specify the service quality to be provided in detail.

- Describe detailed the key performance indicators, metrics and service levels.

- Breakdown transparently all the costs.

2.2.1 SLA Life Cycle

The life cycle of a service level agreement involves several steps for a successful use of SLAs[134]. There are different views on whether the negotiation phase of the SLA is one of its life cycle or not, since this can also be counted among the preconditions (cf. [67] and [59]). Figure 27 shows the SLA life cycle from [121].
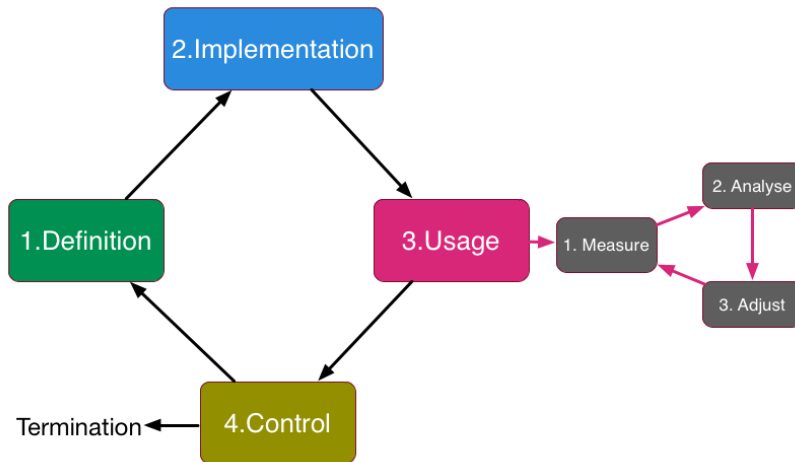
Figure 27: SLA Life Cycle

The preconditions for this life cycle is that the negotiation phase is integrated within the definition, where the deliverable service levels and the costs are negotiated with the provider. This negotiation process usually includes several rounds of proposals and definitions of services.

17

After it is done the agreed on ,binding SLA is created and signed by both parties. While in the implementation phase, the entry into force of the agreement is marked by the signatures of both partners. Here the provided services are provisioned and the agreements are communicated and fitted into the organizations. During the usage phase the customer uses the service according to his notions. Parallel to this, the service performance is monitored during runtime and assessed against the service levels. If needed, corrective actions are executed and reports and documentation are created for the partners. The final control phase marks either the end of the usage by the customer, if he no longer needs or wants to use the service, which initiates the decommission of the service. Otherwise the SLA is checked against the altered requirements and then a new definition process is started.

2.2.2 SLA Content

The structure of service level agreements are generally very scenario specific and can not be easily generalized. However, there are some basic elements that should be present in every SLA. The following remarks are not intended to be used to create an universal pattern for SLAs, but rather give a outline for most current contents of SLAs. The contents of a SLAs can be generally divided into the following four categories: (see [20]) *agreement-related elements, service-related elements, document-related elements and management-related elements.*

The **agreement-related elements** contain the basic rules of the agreement and include, among others, the subject of SLAs, objectives, partners, as well as the scope, entry into force, duration and termination of SLAs. Often these elements are shown in practice in the form of a preamble or introduction. The subject of SLAs introduction here describes the content and context as well as a description and demarcation of the services being controlled by the SLA.The objectives of the SLAs reflect the specific objectives of both parties and serve, among other things, as a basis for future success control.

The **service-related elements** represent those elements which describe the regulation of a service. These must be specified individually for each service. The content is basically to describe who, when, where, and what services are provided. The description of the service should be generally understandable. The description of the quality of a service is the central role of the SLA. The negotiated quality of service is often represented by technical Key Performance Indicators (KPIs), which form the basis for the Service Level Objectives" (SLOs). These indicators include a label next to the calculation or metric, and a reference area and measurement point. KPIs also often target organizational objectives of the service provider and not only technical aspects. It should be noted that not every metrics automatically forms Key Performance Indicator. KPIs are bound to organizational or services goals and must aim towards attainability.

Document-related elements include administrative and editorial elements, which play a minor role inside a SLA and are mainly there to improve the handling, understanding and

```
1. Preamble
   1.1  Subject
   1.2  Goals
2. Partner Description
3. Scope
4. Entry Into Force, Running-time and Termination
5. Service-description
   5.1 Service 'X'
        5.1.1 Contents
              5.1.1.1 Name, Description, Demarcation
              5.1.1.2 Partial Services
              5.1.1.1 Flow, Conditions
        5.1.2 Quality of Service
              5.1.2.1 KPI 'Y'
                    * Name, Description
                    * Metrik, Calculation
                    * Measurement Point, References
                    * Service Level
                    * Reporting
                    * Consequences of Failure
              …

        …

6. Payment and Billing
7. Reporting
8. Consequences of Failure
9. Arrangements to Control the SLA
10. Arrangements to Change the SLA
11. Rules to Resolve Conflicts
12. Privacy and Sercurity
13. Liability and Warranty
14. Compensation, Applicable Law, Jurisdiction
15. Privacy, Confidentiality, Publication
16. Severability Clause
17. Signatures
18. Attachments
```

Figure 28: SLA Structure

readability. These elements are, e.g., version, the date of last modification, revision history, table of contents, the index or glossary. These elements increase the readability by underpinning the context and explain the background.

The **management related elements** include the aspects that have to do with the administration and control of SLAs. These represent a very important section of the contents of a SLA,since both the customer notification and the procedure in case of problems or failures to meet the service levels are regulated. Furthermore, penalties and compensation in case of damage which may occur due to deviations from service levels are regulated.

Based on the presented elements, a generalized structure of a SLA has been created, which can be seen in Figure 41. Here, it is clear that the service descriptions, or service level objectives form the central aspect of each SLA. These and their contents are described in more detail in the following sections. Likewise, it comes clear that even small SLAs mean large administrative overhead and the creation is a lot of work.

2.2.3 Service Level Objectives

Service Level Objectives (SLOs) are a central element of every service level agreements (SLA), which include the negotiated service qualities (service level) and the corresponding Key Performance Indicators. SLOs contain the specific and measurable properties of the service, such as availability, throughput or response time and often consist of combined or composed attributes. SLOs should thereby have the following characteristics: cf. [133]

- achievable / attainable

- repeatable

- measurable

- understandable

- significant

- controllable

- affordable

- mutually acceptable

- influential

A SLOs should always contain a target value or desired service level, a metric and corresponding measurement period, as well as the type and location of the measurement. For this purpose, usually KPIs with associated service level values are stated. KPIs should contain information about the measurement process, measurement destination and unit as well. A valid SLO specification might, for instance, look like this: *The IT system should achieve an availability of 98% over the measurement period of one month. The availability represents thereby the ratio of the time in which the service works with a response time of less than 100ms plus the planned downtime to the total service time, measured at the server itself.* From such a description, the actual performance values can be compared with the reference values of the SLOs and the achieved availability result is calculated. Based on this, further measures can be carried out or correction measures can be conducted if necessary.

2.3 Summary

In this chapter the historic development and essential attributes of cloud computing were introduced, as well as a definition of cloud computing together with a reference architecture were given. The main cloud service and deployment models were presented and related technologies were introduced and delimited from cloud computing. Finally the advantages as well as the

disadvantages of cloud computing were discussed. Additionally, the essential attributes and contents of service level agreements have been stated as well as a definition based on ITIL V2. The main SLA elements were discussed and the related life cycle was introduced and illustrated. Finally a generalized structure and service level objectives were introduced. Along side with the here introduced self-management respectively autonomous systems , this forms the foundations for the system presented .

SLA MANAGEMENT AND THE CLOUD

As mentioned before, SLA contracts describe the exact service quality a user can expect, how fast a provider must response in case of problems and what redress the provider has to give when the SLA contract gets violated and the user suffers a loss of business. SLAs therefore are the cornerstones of every IT service provider to reliable deliver services to his customers. According to a survey by Vanson Bourne Technology Market Research, commissioned by Compuware [139], German companies suffered heavy losses due to poor performance in cloud applications. More than half a million euros, is the average annual loss due to lack of SLAs according to the study. In order to make cloud services effectively usable [58] and reliable for enterprises [64], service level agreements (SLA) are needed, which state the precise level of performance, as well as the manner and the scope of the service provided. To include this functionality into existing cloud infrastructure specialized SLA management is needed. This chapter will introduce the current situation of SLA in cloud computing. Give an overview on related research efforts and introduce identified KPIs for the use withing cloud SLAs.

3.1 Current Cloud SLA Landscape

As the usage of cloud service by companies continues to grow, the need for SLAs is increasing. NIST 92 has pointed out the necessity of SLAs, SLA management, definition of contracts, orientation of monitoring on Service Level Objects (SLOs) and how to enforce them. A basic discussion of SLA management and cloud architectures can be found in Service Level Agreements for Cloud Computing [66], but it is mainly concerned about SLA definitions and negotiations.

At present, most cloud computing providers only offer generic SLA. Thereby guarantees for QoS characteristics like, bandwidth, data backup, etc. are given on the best-effort principle. Companies require QoS, monitoring and control of the cloud services at any time, as stated in the

"Architecture of Managing Clouds" [38] and others (e.g., Study Group Report of Cloud Computing [79]). For cloud computing, the quality and reliability of the services has become an important aspect, as customers have no direct influence on the services. Therefore service level agreements cloud be fundamental to an effective cloud utilization. For the users of cloud services, especially small and medium sized businesses, it would be very desirable to find a cloud provider who can guarantee the quality of the provided services by offering and enforcing SLAs.

Cloud infrastructures offer the potential to enable individual SLA negotiation and enforcement by adapting services during runtime, but this is currently not utilized. In the current cloud landscape large providers such as Amazon are currently not offering customer-specific SLAs and provide their customers with only rudimentary ‚Äùone size fits all‚Äù general agreements. In the case of Amazon, for example, this means a guarantee to all customers of a availability of 99.95% for the EC2 service but for their Elastic Block Store (EBS) no service quality guarantee is given, which can lead to problems.

During the lifecycle of a service the customer is often confronted with alternating and changing demands, which is often reflected in a change of the agreed service quality, emergency procedures, costs, legal compliance, and so on. The classical approach of negotiating SLAs is a very static based process. Unfortunately, this can not keep up with the dynamic character of cloud computing and therefore is unsuitable. For this new methods have to be created the cope with the fast-paced dynamics, but so far there is no solution on this field. Automation in term of SLA management would require SLAs to be presented in a machine readable form. In recent years, a significant amount of research has been performed on the standardization and creation of machine-readable formats. There are two major specification for describing SLAs, WSAL [96] and WS-A [82]. The Web Service Agreement Language (WSAL) [96] was developed by IBM with the focus on performance and availability metrics. It has been mainly developed for Web services and the usage in other fields is questionable. It shows significant shortcomings regarding content as it was focused mainly on technical properties. WS-Agreement (WS-A) [82]. was developed by the Open Grid Forum in 2007. The newest update, which is based on the work of the European SLA@SOI project, was done in 2011. Although it has been enhanced within the SLA@SOI project [2], the development is unclear, because the SLA@SOI project developed its own format SLA(T), which is supported by the European IT industry.

Although much research has been done in the direction of SLA formats, the contents of SLAs remain a further field for investigations. The fact that SLAs are always very scenario specific makes it difficult to generalize their contents. KPIs, as a central component of service level objectives, are increasingly offered in KPI libraries.[122]. However, these are mosty of rudimentary content and are not suitable for implementation. In order to describe the QoS of cloud services, metrics must fit the requirements and orientation of the of the to be measured service. Currently there are no standardized cloud specific KPIs and therefore fitting cloud KPIs standards have to be introduced. Currently the International Organization for Standardization

is working on SLA for cloud computing a definition of SLA content and usable metrics is part of the project ISO/IEC NP 19086-2 [50] and will be publicized in the fourth quarter 2014. This cold lead to a standardized cloud SLA.

In addition, problems can arise dut to the distributed character of the cloud. If for example a breach of contract occurs the jurisdiction may lie outside of the EU and has to be handled internationally. Therefore analysts of the Experton Group [46] recommend german business customers to choose cloud providers with german contracts and service level agreements (SLAs), and local jurisdiction.

Additionally current cloud services are hard to monitor for the customer, because none or only sparse information is given by the providers. However, for businesses, it is essential to monitor their services and check on the compliance with their SLAs. Especially in cloud computing, due to the dynamic cloud character, the QoS attributes must be monitored and managed consistently.[83]

Recently different cloud providers have started offering additional services for their cloud offerings. For example Rackspace[113] is now offering a managed cloud service with three optional service levels (Managed Infrastructure, Managed Operations - SysOps, and Managed Operations - DevOps Automation). These services mainly include arrangements that involve the support and help desk, such as 24x7x365 support systems with different response times or integration and security guidance. However, in the largest configuration, the management of the backup service and the performance by scaling, as well as cloud monitoring is also included. This can be considered as a first step towards SLA integration into cloud services, since thus cloud customers will be given at least some degree of control over the obtained services. However if you look deeper into the offered service, you recognize that no state within respect to the quality of the performance that has to be complied is stated. Likewise, it is not possible to negotiate individual service levels, there are currently only prescribed levels available by the provider. So it remains questionable whether this is sufficient enough to offer these services with enough dynamism and reliably for business users.

3.2 KPIs for Cloud Services

Together with industry partners the ASLAMaaS Project [15] identified over 90 possible cloud service KPIs. The utilized KPIs include, both general QoS parameters such as availability, response time or bandwidth, which are commonly used for almost all services today, but also cloud-specific KPIs, such as the deployment-times for PaaS, virtual image management or scaling schemes. This section will outline the structure of the proposed KPIs and introduce selected sample KPIs. For an complete overview and in detail description see [16].

The cloud KPIs are generally grouped in two domains, the Service Specific KPIs and General Service KPIs. The latter represent all the basic needs of each service has to run efficiently. Service Level Agreements must always be tailored to fit the service that shall be controlled.

Nevertheless, there are some KPIs, which rules can be used in various SLA. These include, for example the availability, security aspects, service times and helpdesk, as well as monitoring and reporting. These are basic requirements for every purchased service. The Service Specific KPIs differ strongly depending on the cloud service model and comprised resources.

3.2.1 General Service KPIs

The General Service KPIs form the basic requirements every cloud service bares. They include the availability which is defined at the time the service is usable, the Mean Time Between Failure and Mean Time To Repair, which specify the time intervals at which to expect failures and how long it takes to repair them. Additional security KPIs regulate for example which software version levels shall be used, how long it should take until an update is implemented, as well as the scope and frequency of security audits. Other important KPIs control the encryption of data, the use and timeliness of anti virus software and the isolation and logging. Service and Helpdesk KPIs control the times at which assistance is provided, which support methods are applied or how many calls are received per week. Similarly, the qualification of the support personnel and the duration is given to problem solving. Finally monitoring and reporting KPIs are used to define in which determined intervals performance date id to monitor and how to handle the resulting reports.

A sample KPI definition out of this group would be the availability, which is composed as follows: Availability of an IT system, the status of the error-free usage of the functionality of a system under specified conditions within a defined time frame. The error-free usage refers to the defined functionality of a system, such as sending of e-mails, so the service would already be unavailable if no e-mails can be sent, even though the system is reachable and the receipt of e-mails would be still possible. In practice, is the fact that services are switched off at certain times to perform, for example, maintenance or updates needs to be considered. Therefore, the availability is calculated as follows:

$$Availability = \frac{Servicetime + Planed\,Downtime - Unplaned\,Downtime}{Total\,Time}$$

The Servicetime here refers to the time in which a system could be used without errors. This is also often referred to as operational time. The downtime is the non error-free time. A distinction is made between planned and unplanned downtime. As planned downtime refers to maintenance, updates, and so on, which has been agreed on in advance with the customer. The total time indicates the reference period for the measurement in which this calculation is conducted. The benefits of such agreement is strongly dependent on the definition of the reference period. For example, an availability of 98.5% guaranteed for a reference period of one week results in a permissible outage of 2.52 hours per week. However, if the reference period is set to one year the allowed downtime would be about 5.5 days (131.4 hours) per year. In practice, the availability

is usually provided in relation to one year. Another critical factor is the definition of the type of measurement. It is important to note that the wanted functionality of the system has to be imaged. For example, a simple ping or a request-response time below a certain limit can be considered "available". In complex distributed system there is a additional requirement that it hast to be determined whether the availability of a individual component equals the aggregated system overall availability.

3.2.2 Service Specific KPIs

The service-specific KPIs are based directly on the service model of the services and its resources. Particularly for cloud computing, the network has a strong meaning, as all provided resources and services are available through a network. Here, the **network** has to be considered both as a pure transmission medium for other services as well as independent service itself. For the described KPIs, the entry point of the provider network is usually chosen as a measure point, as the guarantees of the provider refer only to this area of the network. A sample KPI form the network group involves the *Round Trip Time*, *Response Time*, *Packet Loss*, as well as *Bandwidth* and *Throughput*. These are sole technical parameters.

Cloud Storage KPIs can be distinguished within cloud computing in two basic types. First, Storage as a service itself, that is obtained as a memory for preexisting infrastructures. On the other hand storage can be used as part of another service such as a backup or data storage for cloud services. Typical KPIs here are the *Response Time* being the interval between sending a request to the storage and the arrival of the response at the output interface. Usually measured in milliseconds. The *Throughput:* Number of transmitted data per time unit. Here, a specified amount of data is transferred to the storage and measured the needed time from a given point. The size of the data set and package sizes are important factors for the validity of this measure. Furthermore, the network and its utilization must be considered. The *Average Read/Wirte Speeds*, which in contrast to the throughput, refers to an individual hard drive or hardware type. This value indicates how fast data can be read or written from/to the hardware. In RAID systems or virtual storage solutions, this figure is expected of interconnected hard drives. *Random and Squential Input / Outputs per second (IOPS)*, which give the number of possible random / sequential input / output operations per second for different block sizes. The higher the IOPS value, the faster the disk. This value is also important to measure how many concurrent accesses can be handled by a system. More cloud relevant KPIs are the *Provisioning Type* and the *Average Provisioning Time*. The type of provisioning where at "thin provisioning" the client gets the storage not permanently assigned but it is dynamically allocated at runtime. In contrast, the thick-provisioned storage is allocated to the customer immediately and the time, the provider needs to provide a defined amount of data volume growth.

Backup and Restore KPIs refer to both the storage, i.e., the stored data, as well as services, for example, VMs or SaaS services. Here the *Backup Interval* is a major KPI together with the

Backup Type . An exact specification along with the backup type and a description of the scoop has to be given to to the provider in order to protect from data loss. *Time To Recovery* specifies the minimum and maximum time from the failure of a storage, to the successful restore from an existing backups. Together with the *Backup Media* and *Backup Archive* they form a complete definition for cloud services.

Depending on the service model **Infrastructure as a Service KPIs** refers not only to the service itself but also to the virtual machines used. For this, additional VM KPIs are specified as well. For infrastructures the *VM CPUs*, meaning the number and type of CPUs used by the virtual machine hast to be defined. Additionally information about the overbooking of the provided CPU resources shall be given. Here the shared resources are allocated with more capacity than is physically available. Thus, no real physical allocation of resources takes place. Actual performance is dependent on the overall consumption of the system. The KPI *CPU Utilization* enables performance management by being based on the proportion of CPU resources in use to the total number of resources provided per time unit. Also the CPU queue, which indicates the number of open requests to the CPU should be considered. Similar KPIs for the memory exist with *VM Memory* and *Memory Utilization* likewise here information about the overbooking of allocated memory resources should be stated. Cloud service type specific KPIs like the *Migration Time* and the corresponding *Migration Interruption Time* define the maximum time in which a customer has no access to a resource while migration. *Logging* gives retention of log data, which specifies how long log data to be stored by the provider and specification of what level to be logged. (e.g., INFO, DEBUG, etc.)

These KPIs presented here represent a small selection from the wealth of the identified potential cloud KPIs. However, the selection of a service-related and the specific management can not be generalized.

3.3   QoS Control in the Cloud

In order to guarantee the QoS in a SLA a provider must be able to control the respective QoS parameters. Otherwise a provider can only measure the performance of the service he offers but can not actively adjust its quality, which makes the offering of SLAs outrageously. The control of the relevant QoS parameters always requires the adaptation of the underlying infrastructures or its resources. So for example, the improvement of CPU utilization requires either the control over the virtual CPU resources provided or to the entire infrastructure. In the first case a simple alteration of the assigned virtual CPUs can change the utilization of the entire system. In the other case the overall utilization could be altered by adding additional VM instances which then get the occurring load distributed by a load-balancer. In both cases the service of the cloud user does not notice an continuously keeps running.

First research in the area of QoC control has been carried out during the first phase of this

research [55]. The research involved a load-balancing scenario in which there was a response-time SLA. The number of virtual instances which could be used parallel was decided based on this parameter in order to minimize SLA violations. Currently cloud provider like Amazon offers services like Amazon Auto Scaling, in which a user can define rule sets on which the number of EC2 instance gets altered. In the conducted research this type of control mechanism was evaluated against fuzzy rules. Different load scenarios and additional simple prediction model have been introduced.
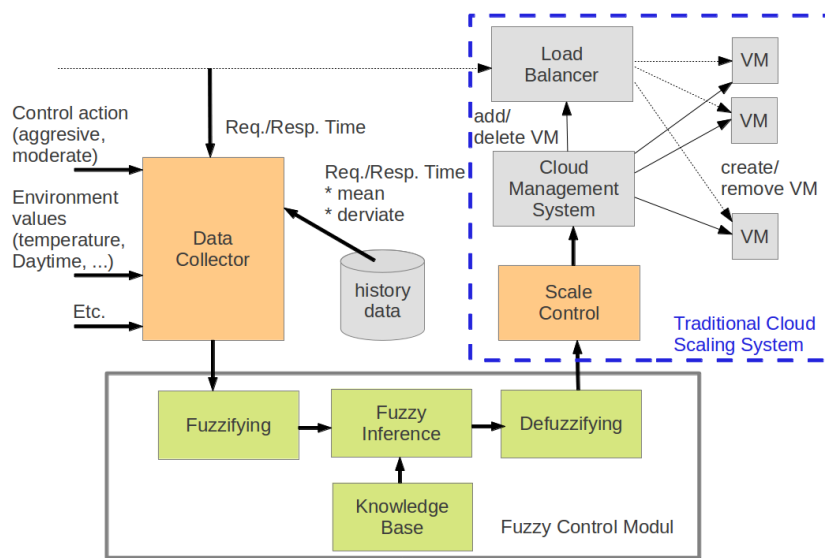
Figure 31: QoS Fuzzy Control Architecture from [55]

The aim of the research was to show how a common cloud computing scaling service could be enabled to guarantee QoS parameters. Figure 31 shows the proposed architecture for enabling QoS control by Fuzzy rules. As proof of concept several test were performed in order to prove the effectiveness of each approach. Results showed that this method of controlling a cloud QoS parameter, in this particular case the request-response time, can result in significant less SLA violations. This marks a first step towards QoS control in the cloud, but additional parameters have to be tested on whether or not and by which methods they can be adjusted.

3.4 Summary

In this chapter the current Cloud SLA landscape and developments towards SLA for cloud services have been presented. The general SLA management process and its adaption to the cloud computing model has bee presented. First insights and research results regarding KPIs

usable for cloud SLA have been outlined. Finally already carried out test regarding the control of QoS parameters fo cloud services have been demonstrated.

Autonomic Cloud Computing SLA Management Approach

4.1 Autonomic Computing

The ever increasing complexity, multi dimensionality and abstraction of modern computer systems make it difficult for users to utilize, maintain and fully understand them. With this issue in mind, IBM [69] developed a concept for autonomous systems and coined the phrase Autonomic Computing. According to the IBM‚Äôs vision autonomic behaviour can be characterised by key capabilities such as self-diagnosis, self-conÔ¨Åguration, self-adaption and self-healing, which are all focusing on the continuous executability of a computer system without the need of user interaction. There are sveral definitions for autonomic computing, following the most common ones are stated:

- ”The vision of autonomic computing is to create software through self-* properties” [132].

- ”Autonomic computing is the ability to manage your computing enterprise through hardware and software that automatically and dynamically responds to the requirements of your business” [102].

- ”Autonomic computing is the ability to manage your computing enterprise through hardware and software that automatically and dynamically responds to the requirements of your business” [102].

The phrase autonomic is derived from human biology, where the autonomic nervous system keeps track of the bodies vital functions such as heartbeat, temperature and breathing while working in the background. However, there are main differences between the human body and autonomic systems. Firstly most of the tasks of the autonomic nervous system can’t be controlled,

where as the tasks of an autonomic systems are based on human written policies. To make these policies work the system has the need for sensing the state of the system and its enviornment. According to the IBM vison, a autonomic system consists many interacting autonomic elements. These elements control the autonomic computing environment, consisting of other autonomic elements or the managed system, based on defined policies. This was modeled into the autonomic control loop (MAPE-K paradigm).
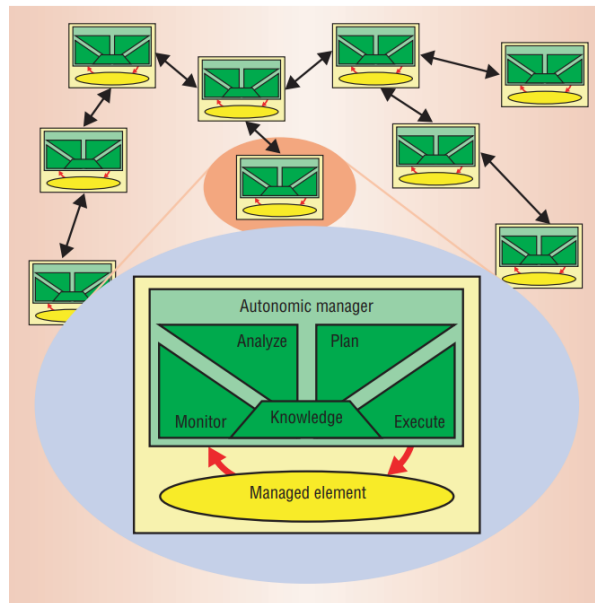
Figure 41: Autonomic Control Loop (MAPE-K) after [84]

The four phases, monitoring, analysis, planning and execution are extendend by a common knowledge base and form the so called autonomic manager, which directly interacts with the managed element. Such managed elements can consists of, for example CPU, network nodes, virtual machine instances, databases or applications.

- **Monitoring** is collecting informations and metadata about the status, process and trend of the managed element. Here various methods of monitoring can be used, such as push or pull systems or agent-based systems to periodically retrive the needed information.

- **Analysis** uses these collected informations to decide whether or not the indicaton of an event to start an adaptation action has been reached. If such an event occurs an action strategy should be created to adapt and adjust the system to the targeted state.

- **Planning** uses this action strategy to create an adaption plan based on the targeted state and the current state of the system. This plan consists of instructions, wich in detail describe the actual changes and actions.

- **Execution** runs these adaptations and adjustments. Herefore the execution of the system is altered and the adjustments are made. Afterwards the loop starts again.

The Dynaco adaptation model proposes a similar approach to the MAPE for distributed application by leveraging component-based design [25] [24]. Autonomic computing aims to faciliate self-management of complex systems consistiong of various components. Autonomic computing systems involve service-oriented technologies, agent technologies, adaptive control theory, machinelearning, optimization theory, and many more [48] [148].

The aim of introducing this paradigm in this thesis, is to achieve autonomic behaviour in our proposed Cloud management infrastructure, whereby appropriate actions are taken towards the enforcement of SLAs and performance metrics.

This report has identified the requirement for cloud computing infrastructures to guarantee service qualities and the need for new mechanisms for the integration and management of service level agreements.

However, to address the identified problems within the current cloud computing landscape a system is needed which supports dynamic and fine grained quality assurance, while being flexible enough to cope with the frequently changing infrastructure. And automation of the underlying SLA management process, which includes creation, monitoring, reporting, control and adaption. Especially for this more research in the area of behavior analysis, prediction and anomaly detection is necessary.

4.2 Autonomic SLA Management Architecture

To empower cloud users with the possibility of managed cloud service quality and simultaneously increase the trust, reliability and appear of cloud computing for business use, this research proposes the concept of autonomic continuous dynamic SLA management for clouds, which will be realized by the ‚ÄúAutonomic SLA Management as a Service (ASLAMaaS)‚Äù architecture. It is build upon the IBM developed autonomic manager concept MAPE-K [97] (Monitor-Analysis-Plan-Execute-Knowledge).

During the first phase of the research, cloud specific issues regarding the quality of cloud services and the need for dynamic management of service qualities were identified and analyzed, see Chapter 4. The ASLAMaaS framework is targeted on the following cloud challenges:

The goals of the presented ASLAMaaS architecture are to integrate dynamic, autonomous SLA management into existing cloud infrastructures to enable cloud users to independently conclude SLAs for cloud services on a self service basis to improve the overall service quality and reliability, as well as increase the trust and economic use of cloud computing in general. For this the architecture shall enable continuous and flexible monitoring for cloud resources in regard of the corresponding service levels. Additionally advanced reporting and logging is integrated in
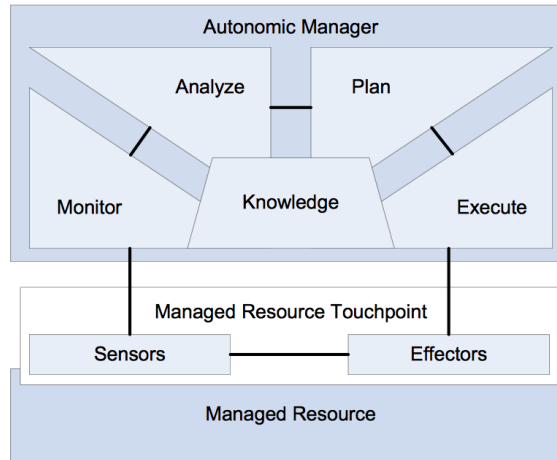
Figure 42: Autonomic Computing Control Loop from [97]

every module of the architecture to enhance transparency between service provider and consumer by enabling comprehensibility.

Due to the strong dynamic and rapidly changing character of cloud computing, it is necessary that the system continuously adapts. Automomic computing systems are capable of continuous self-monitoring and adjustment. For this reason, the proposed architecture is based on autonomous modules which operate according to the principle MAPE-K. The overview of the components is shown in Figure 42.
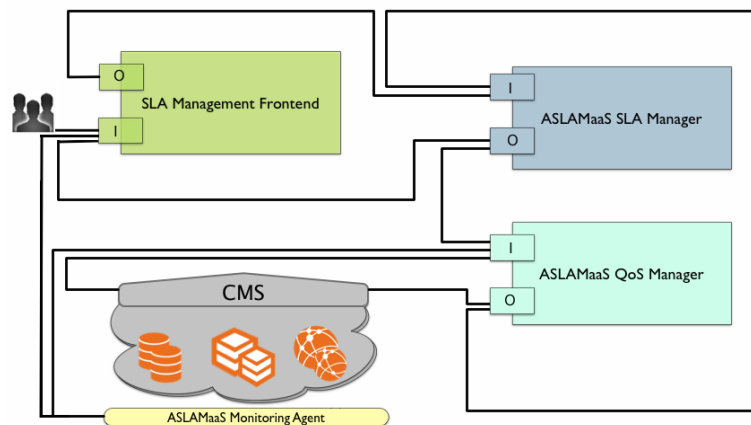
Figure 43: Information Flow between ASLAMaaS Modules

For the monitoring of the autonomic system, a sensor is connected to the managed resource touch point, there the monitoring unit collects the data from the sensors and hands it over to the analysis unit, which then with the help of an external knowledge base creates action plans. These plans are carried out by the implementation unit (Execute), which adjusts the managed resource

through effectors. In the case of a cloud infrastructure this could be upscaling the amount of CPUs of a VM instance.

The presented architecture consists of three main modules, the SLA Management Frontend, SLA Manager and the QoS Manager. The information flow between the individual modules can be seen in Figure 43. The SLA Management Frontend thereby marks the input layer of the system, where users can create, monitor and adjust SLAs for their services.
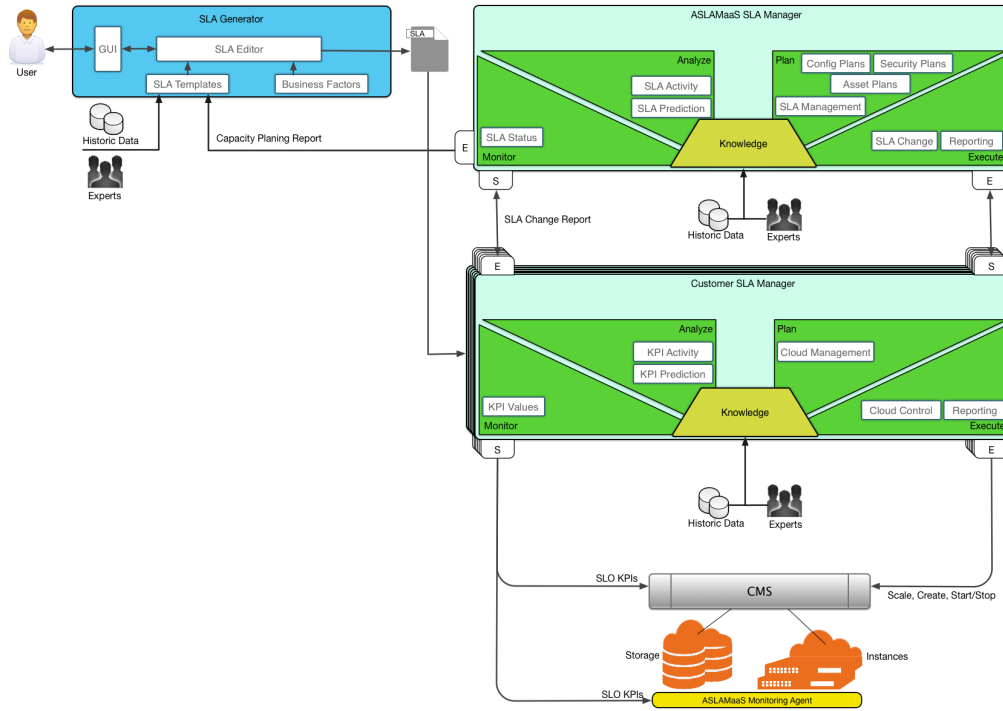
Figure 44: ASLAMaaS Architecture Overview

To facilitate the process of creating new SLAs there will be a repository of pre-made SLA templates available within an graphical SLA editor. Thus users can easier and faster create their own specific SLAs by selecting a fitting template, which shall reflect best-practice and experience data for generalized types of services such as web services, databases systems, ERP systems and so on. The used SLA templates will use different key performance idicators (KPIs) grouped into categories based on their usage type. For each KPIs a specific QoS parameter and the associated metric has to be stored within the repository. A more detailed outlook on the SLA creation process and its components can be found in subsequent Chapter 4.3.

After the creation and activation of a SLA for a cloud services the SLA Manager stores it in a repository and starts the autonomous control loop. An overview of the ASLAMaaS SLA Manager and its components is shown in Figure 45. The SLA Manager is used as an abstraction level, in

which the individual SLAs and their dependencies between each other are processed. Within the SLA Manager the general functionality regarding SLA compliance is processed.
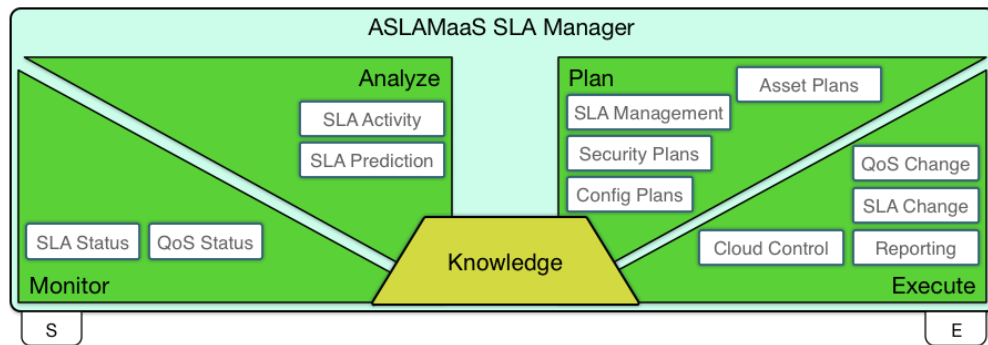
Figure 45: ASLAMaaS SLA Manager MAPE-K Loop

For this measuring and keeping track of the status of each SLA and the containing service levels is needed. Therefore, the cloud management system, virtual resources through intelligent agents and archived data will be used as input sources. Within the Analyze part the data is used to perceive the current state and predict the future behavior of the managed SLAs. Here the mutual influencing SLAs factors and the strategic business planing of the provider shall be weighed against each other. This could mean, for example, that in a state of emergency some contracts could be abandoned in favor of others so that the resulting financial damage will be reduced. Likewise here the beginning, suspension and stop of the dynamic SLAs and the total infrastructure overview data shall be used for strategic planing and pricing.

Depending on the specific service, cloud SLAs may consist of several independent or conditional KPIs In order to ensure the service quality and operate within the agreed on limitations every KPI has to be controlled and monitored individually. For this every managed service level respectively KPI within a SLA will get its independent autonomous management loop. The corresponding graphical overview is shown in Figure 46 below.
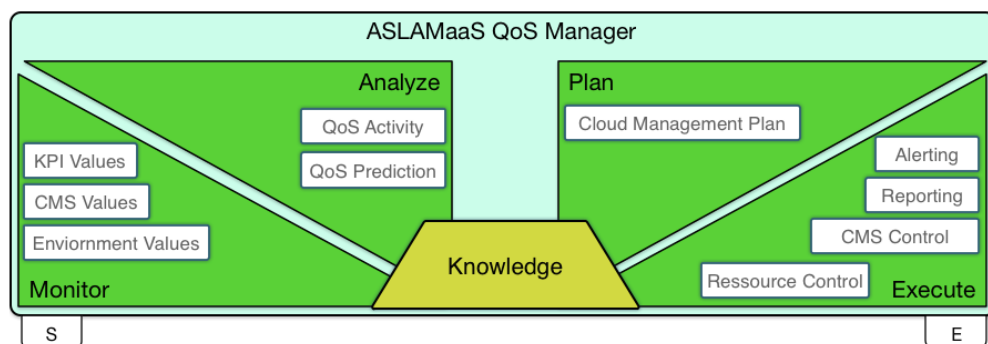
Figure 46: ASLAMaaS QoS Manager MAPE-K Loop

The basic functionality will be nearly the same as the administration by the SLA Manager, only that instead of SLAs the individual QoS characteristics are directly influenced by adjustment of the cloud infrastructure and resources. So every SLA Manager instance will have multiple instances of the QoS Manager wich will take control of the corresponding parameters, monitoring and reporting. Starting by the Monitor, KPI relevant technical parameters, environmental data and dependencies are collected. This will be done by using the stored metrics for each KPI, deposited within the Knowledgebase. Such metrics will be measured either directly within the Cloud Management System (CMS) or directly on the resources by using intelligent monitoring agents. Within the Analyze part prediction algorithms are used to estimate future problems and estimate long term developments. Based on these calculations, respectively the current state of the monitored parameters, action Plans will be conceived. Such plans will consist of actions like scaling up or down, allocation or deallocation of resources, altering the infrastructure, or alerting the corresponding service provider if an adaption is not automatically possible.

The proposed infrastructure shall intervene directly with the CMS. As reporting is an important core component of SLA management, for example as evidence in case of SLA violations or as statement of the delivered service level, the continuous reporting and logging must be ensured. The system has to exhibit to the users that the in a SLA agreed service levels have been consistently met, or remained within the defined deviations. Therefore the reporting is integrated in the Execute part of the QoS Manager as well as in the SLA Manager of the ASLAMaaS architecture.

4.3 SLA Creation Process

To keep up with the dynamic ephemeral character of cloud computing SLA management needs to be adaptable any time. For this classical process of SLA negotiation must be adapted towards the more dynamic self service. In the proposed architecture this task should be solved by SLA Management Frontend. A structural overview is shown in Figure 47.
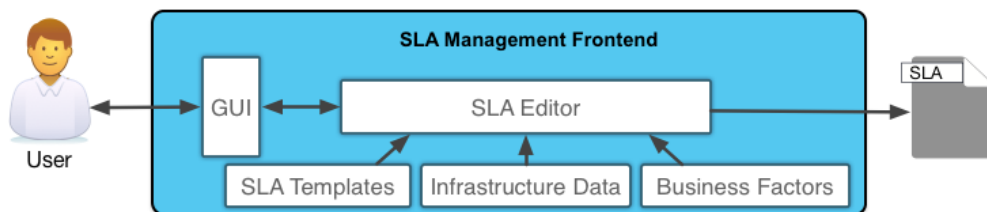
Figure 47: ASLAMaaS SLA Management Frontend Overview

As described in Chapter **??**, SLAs comprise the expected quality of service the provider has to deliver, which are recorded as the agreed service levels. The service levels include one or multiple KPIs, which describe in each case a specific QoS parameter and the associated metric to monitor.

In order to allow users to choose the KPIs for their SLA contract self-reliant certain preconditions must be met. Since cloud architectures are constantly changing the available KPIs are provider dependent and must be pre-accepted beforehand. Therefore possible cloud relevant KPIs have been identified in Chapter 3.2 and must be selected initially by the provider.

Based on the provider accepted KPIs meaningful margins have to be calculated as basis for the SLA creation process. This is necessary because in this self-service approach the provider shall not have to engage in the process respectively negotiate with the consumer. The margins mark the boundaries in which a user can choose the service level. In order to provide only feasible conditions of which the user cam choose pre-calculations have to be made and pre-conditions have to be checked. For this purpose, various mathematical functions shall be examined for their suitability and expressiveness in order to integrate a suitable model.

Additionally the utilization, expected usage and performance data of the infrastructure shall be matched together with business concerns and the strategic focus of the provider to create a dynamic pricing model for the offered services and the corresponding SLAs. If, for example an infrastructure constantly delivers a response-time of below 170ms and only a few fluctuations of users are expected the reasoning model may set the lower margin for this KPI to 200ms or slightly above. Since it is common practice to make prices not directly on each possible service level, but to create pricing categories, such as low, medium, high or silver, gold platinum the delivered price model shall then create these pricing categories. This should enable a simpler management of the managed service levels and makes the gradation of customers easier for the system.

In order to autonomously conduct this process the SLA templates and the finished filled out SLAs used must be represented in a machine processable form. For this a machine readable agreement description language shall be used. The Adaptable Service Level Objective Agreement (A-SLO-A) model [56], which is based on the SLA* model, enables the use of dynamic and constant agreement alteration and therefore is will be used as the basis of the description of SLAs in ASLAMaaS.

4.4 Prediction, Detection and Management of Service Levels

In order to guarantee the QoS of a provided service, it is necessary for particular things to be monitored. Firstly, the performance of the infrastructure must be monitored since dependencies arise in a multitenancy environment with shared resources. Secondly, the use of cloud services usually varies greatly and thus the need of resources fluctuates likewise. To cope with this changing needs and to achieve SLA compliance the provided resources must be adjusted. If one could predict the usage of a service, looking ahead further than the infrastructure provisioning delay time, one could guarantee the QoS for that specific service. Therefore prediction and detection techniques should be

To detect unforeseen events, the cloud components behavior is constantly monitored and analyzed. Therefore events from each of the CMS and monitoring agents are processed to enable anomalous behavior detection. To determine, which anomaly detection algorithms are suited best, an evaluation has to be done. The algorithms considered include machine learning approaches like neural networks or Markov models, statistical methods, outlier detection and some specialized algorithms [106].

The Plan part of the MAPE-K concept in this module creates the action plans fot the infrastructure, the CMS, the SLA Manager itself and additionally delivers adjustments directly back into the SLA Editor, where for example the margins could be readjusted to cope with the changes. Thus this the SLA templates and the corresponding KPI margins are always complaint with the actual performance of the cloud system. The action plans result in adaptation of the cloud infrastructure, like for example if the stated response-time inside a certain SLA tends to be broken the SLA manager can instruct the virtual network agent to re-route the traffic if the problem is network dependent. In case of this problem related to overloaded CPU or virtual instances the SLA Manager could start new instances or allocate more CPU cores.

The various methods to manage the QoS parameters have to be model individually and tested about their suitability. A sample for the regulation of the KPI response-time can be found at Frey et al. [54]. There scalable cloud services have been started and stopped based on a fuzzy control set. This and other similar control mechanisms enable the Execute part to adapt the infrastructure and the services so that SLA violations can be avoided and the QoS is guaranteed. Within both MAPE-K loops the Knowledge consist mainly of the historic data about the services, the SLAs and the cloud environment, which was measured continuously and expert knowledge in the form of best practices and empirical values as well as strategic business plans.

4.5 Evaluation of ASLAMaaS

During and following the development of the ASLAMaaS prototype, the sub and complete system will be evaluated due to its performance and SLA compliance. For performance, experiments with increasing amounts of VMs (10,50,100,...) will be developed to show that the ASLAMaaS approach is scalable and can cope with rising dependencies. Further investigation will be done to clarify the following questions: How accurate is the generation of SLA templates based on the corresponding infrastructure data for the margins? How well is the prediction suited to foresee mutual influences of VM performance? Can the QoS of the cloud services and the reliability be improved beyond business requirements ?

4.6 Summary

The proposed SLA management framework for cloud computing is a novel and robust specification, for supporting QoS and SLAs management in cloud infrastructures. It is designed to address

the growing demand on QoS and performance guarantees arising from the increasing use of cloud computing by small and medium sized industries. Overall it aims to increase the reliability and transparency of cloud infrastructures, giving users the opportunity to manage their quality needs. At worst it is providing the same level of quality management currently found in cloud computing environments. At best, all cloud customers can benefit from it, by defining and changing on-demand the service levels for their cloud instances. SLA reports enable users to get a comprehensible knowledge about the status of their cloud instances at any time and enable continuous verifiable proof about the delivered services. Cloud providers will be able to estimate the potential performance of their infrastructures and based on that give customers service guarantees. Performance evaluation, prediction and behavior analysis is used to detect possible risks and intervene avert an SLA violations. The ASLAMaaS framework is based on concurrent autonomous management modules, which evaluate the performance levels of cloud resources such as, cloud instances, storage, network and the cloud infrastructure as well as specific cloud services. This will improve the overall reliability and performance of cloud infrastructures.

This chapter discusses the individual components and the resulting test evaluation.

5.1 SLA Frontend

5.1.1 KPIs and Infrastructure Data

As introduced in chapter 3.2, KPIs are widely used to measure and overview computing environments. In this implementation and evaluation environment data has been collected via the Ganglia [60] distributed monitoring system, which is based on a hierarchical design and targeted at federations of clusters. Figure 51 below shows the Ganglia interface with collected infrastructure data from our OpenStack Cloud environment.

Here it can be seen how core KPIs, such as memory usage, cpu load, process load and network traffic is monitored. The test environment consisted of two worker nodes. Here different types of load scenarios where used by running an request generator, which accessed the services. This data was then collected and archived during a 6 months period.

Figure 52 above shows the aggregated load graph for the cluster during the testing period. All data was as well collected in a time series data store. Additional in the scenarios described in later sections, simulated workloads have been used to generate a repeatable and comparable environment. Various typical workload scenarios were depicted, such as a burst load scenario, which could also be observed on the live system due to real life usage.

5.1.2 GUI

In order to interact with the implementation a GUI was designed to facilitate the use and the feel of the prototypical system.
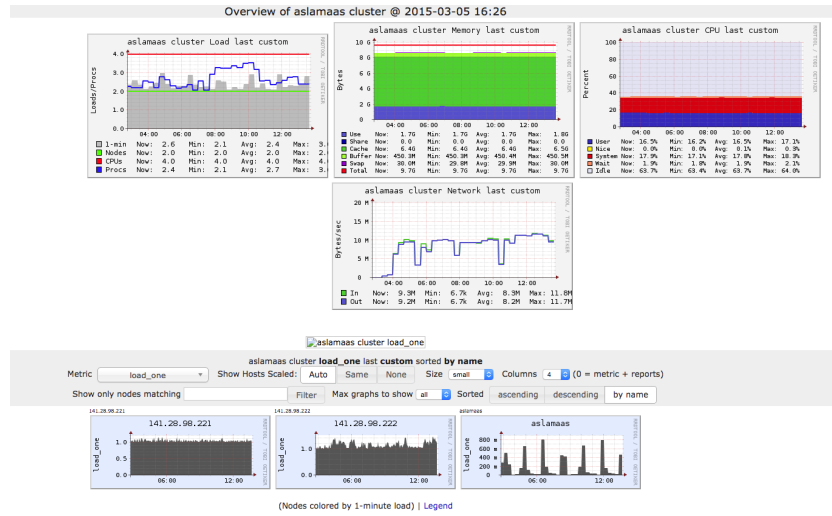
Figure 51: Ganglia Cluster Overview
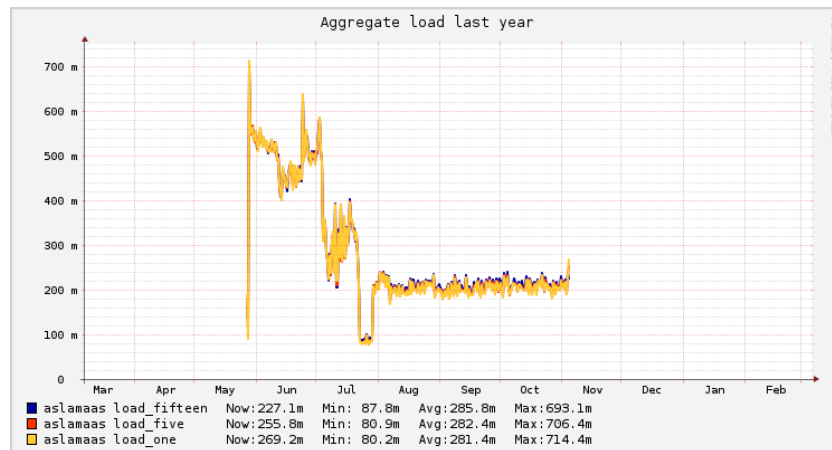
Figure 52: Cluster Load Aggregation

5.1.3 A-SLO-A

The reference implementation of the SLO-A format is based on the abstract SLA* model, developed within the SLA@SOI project citeKearney2011b. The SLA* model is an abstraction layer and follows the description of the Meta Object Facility (MOF), specified by the Object Management Group (OMG). MOF describes a special meta data architecture and itself uses the highest abstraction layer M3. The SLA* model is on layer M2, this correspondences to the Platform Specific Model (PIM). The SLO-A Format is one abstraction layer below (M2). This describes the Platform Specific Model (PSM). So the SLO-A Format is on the same level like SLA(T) description of the SLA@SOI project, as shown in Figure 56. For the reference model of SLO-A it has not be used the primitive data types of the SLA* model instead the EMF data types of the Eclipse module Ecore are used. This was necessary to get a functional prototype.
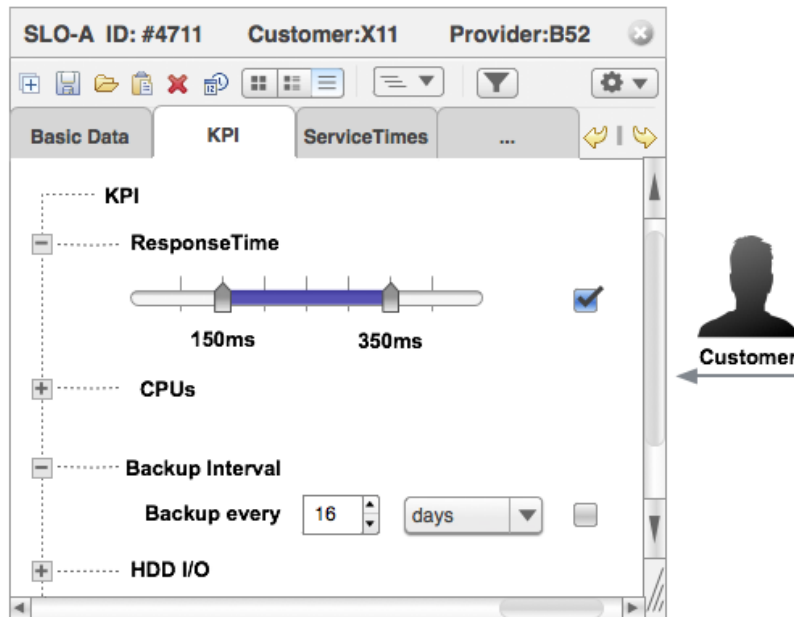
Figure 53: First GUI Mockup
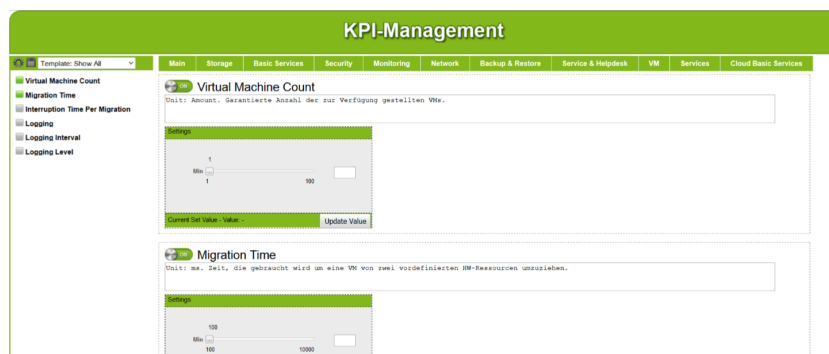
Figure 54: Final GUI

The main goal of the format is to develop an adaptable, adequate machine readable agreement which is legally binding. To get customer-oriented, runtime-adaptable SLAs it is first divided into a static and a dynamic part. The static part, like contract partner IDs, addresses, etc. is important, but more interesting is the part changeable during runtime, focused on SLOs like max. scaling limit, backup period, etc. This dynamic SLA part must be monitor-able and controlled by the customer.

Within the SLO-A format it is possible to define Top-Level SLAs and SLO-Agreements (SLO-As) which are single and dedicated for each SLO.

Figure 55: SLA Violation Report
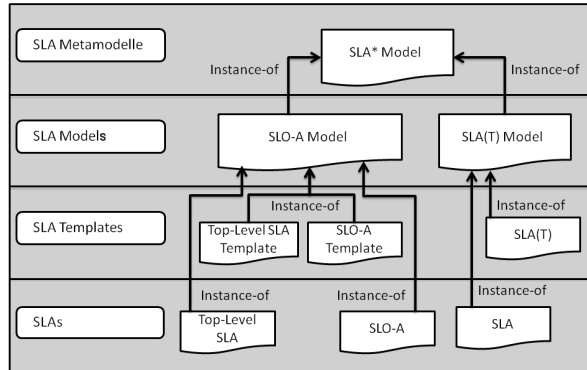
Figure 56: Model Based Development of SLO-A

5.1.3.1 Top-Level-SLA to a SLO

All Top-Level SLAs and SLO-Agreements are based on a appropriated template (see Fig. 57. Also a Top-Level SLA always is displayed by a service which can have one or more interfaces, which are the base of the agreement. On the contrary a SLO-A is not based directly on a service only with the interfaces. Here technically or contractually responsibilities effect can be modelled.

Resulting characteristics of the model:

- Technical services are characterized by a one to one assignment from a Top-Level-SLA to
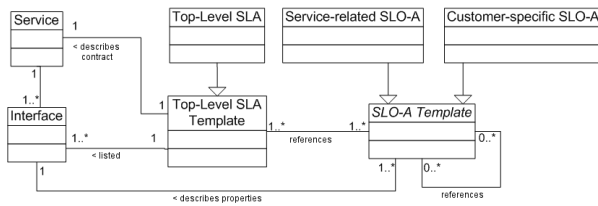
Figure 57: Overview of SLO-A Format

an SLO template, it's a so called service oriented SLO template. Also SLO-A templates can be referenced to another SLO-A template and build a hierarchy.

- SLO-As can be grouped by a higher SLO-A or a Top-Level SLA. This builds a hierarchical tree where each entity is referenced to the next higher SOL-A with an UUID and a naming convention. So a an entity directly references the low SLO-A and monitoring can be done over all.
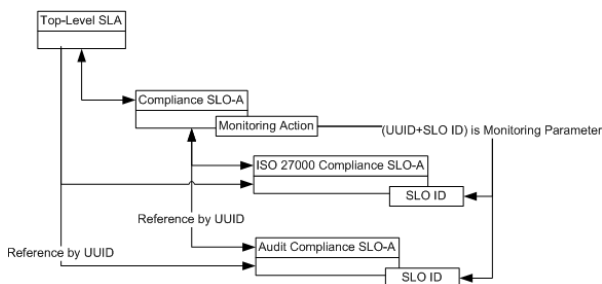
Figure 58: Example of References within SLO-A

5.1.3.2 Top-Level SLAs and Top-Level SLA Templates

Basically the Top-Level SLA comprehend static contractual informations and only few dynamic informations. Also there can be made references to outsourced documents like general terms and conditions. A Top-Level SLA necessary must have informations about accounting of services, IT continuity plan, service development plan, terminology, escalation plans, guidelines for priorities, responsibilities of both customer and company, data of both parties, service times, accomplishment penalties, signing, an ID, how reports have to been done and how the were displayed in which frequenz, the interfaces with ID and description, references of SLO-as with UUID and naming convention, and the termination reason.

For a legally binding SLO-A have to to contain a SLO-A identifier, contact data, service time. Also the SLO itself with an ID, a value and data type which has exactly one interface, which priority, if needed the allocation to other SLO with their interfaces, the boundary an marginal values. The SLO-A additionally should contain the SLO-A reference, the accounting,

accomplishment penalties, monitoring, how and with with interface it is done, how the reporting is done and the termination clause.

5.1.3.3 Workflow for a Top-Level Sla or SLO-A

To generate a Top-Level SLA or a SLO-A , the customer first has to fill in the customer data, servie times, how it will be payed and how it should be reported with which interfaces. In comparison to a SLO-A there also has to be filled in how it should be monitored. After that a decision is made if its an standard template or a customer oriented template. In order its standard all information will be checked and if necessary it is asked for correction. Otherwise if its an customer oriented template, it is asked for a special template. If the value verification is accepted at the Top-Level SLA, all SLO-A are filled in and the inferior references are build. At the SLO-A template the reference is to the higher entity is build directly. After that the templates got signed and the contract are ready.

The incident an change management are mapped as a inferior grouped SLO-A, so they can be used as KPI. An extension of conditions and modality can involve more actions, which always have a condition, guidelines and postcondition. They describe how the action will be triggered.

5.1.3.4 Reference Implementation

The SLA Format implementation is done as part of master thesis by Ralf Teckelmann at the HS Furtwangen University [137]. There have been made many extensions to the SLA Templates proposed by the SLA@SOI project. One important part of a SLA Template (see Figure 59) is the extension of the attribute *Type*. It differs from the Type of the SLA Template by the possibility to have 3 values: *top-level*, *service* and *customer*. This causes an 'IF' clause to load a concrete structure into the SLA template or SLA. So that means, when *Type* is either *service* or *customer* it is a SLO-A Template or SLO-A. For the unique identification the *UUID* attribute is used. Also the model version can be found in the attribute *modelVersion*. The next subsections discuss the main features of the SLO-A Model in detail.

5.1.3.5 SLAs and SLA Templates

As described in last chapter the documentation is as following: There are two segments, that contains the documentation part of the SLA/SLO-A Templates. The first segment *descriptions* keeps the general terms of the agreement. The second segment *fuDocs* (shortcut for further Documents) contains the interface description. Both structures will be described later. Normally it is possible to use SLA and SLA Templates without agreement term segment, but for compatibility reasons to SLA* respectively SLA(T) this segment is included.
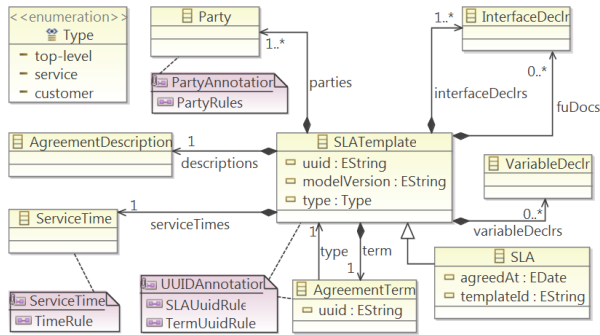
Figure 59: Overview of SLA Template, a modified and extended SLA(T) model [**?**]

5.1.3.6 Description of Agreements

The class *Agreement Description* as segment *description* of SLA templates and SLA has also its own structure. This is a different to the SLA* model. In SLA(T) there exists only a *ServiceDescription* segment without further information. In SLO-A Format the segment *descriptions* contains exactly one element of the class *Property* with its segment *entries*. These *entries* use the class *Entry* with its two attributes *key* and *value*. The attribute *key* contains the type of the following document in the attribute *value*. The type can be either a *CoverageDescr*, a *AgrementDescr* or a *Disclaimer*. The information itself is in the attribute *value*.

5.1.3.7 Time Frames

The next segment of the class *SLATemplate* is the *serviceTimes*. It describes the time the service must be available. The segment uses the class *ServiceTime* one time. All times are save there. This is also a different to the implementation in the SLA* model. In the SLA* model only a start and a stop time defined. This is not enough. So the SLA-O Format creates the class *ServiceTime* which holds in its segment *serviceTimePairs* a list of different time intervals, where an interval contains a pair of a class *entry*, where the time definition is stored.

5.1.3.8 Parties

The contracting parties will be placed in the segment *parties* of the class *SLATemplate*. Here are used the original class *party* of the SLA* model. It contains a contact point with all required information. Also the class *STND* is included which holds information about the *agreementRole*. This is normally either the *provider* or the *consumer*. Also it is possible to store *Operatives*. These are contact persons, which can be directly assosiated to *Actions*. The annotation *PartyRules* describes conditions when information be available in a SLA or SLA template, this is a deviation to the SLA* model.

5.1.3.9 Signatures in SLAs and SLA Templates

A complete new attribute is in the class *SLATemplate* the segment *ProviderSignature* and in the class *SLATemplate* the segment *customerSignature*. This contains the signature of the corresponding party. This feature allows it, to sign a SLA online. Is a SLA accepted by the customer and/or the provider, the corresponding signature will be added to the *SLA* (segment *customerSig*) and *SLATemplate* (segment *providerSig*).

5.1.3.10 Interface Declaration

The interface declaration is part of the service description and is used to connect the interfaces of a service and the SLOs. Also for the reference of further documents it is needed. There exists two ways to implement that, but both uses the abstract class *interface*. First is done by the resource oriented interface *ResVersion*. This is used to keep further external documentation in the segment *endpoints*. The class *Endpoints* contains the three attributes *location*, *id* and *protocol*, to describe how to access the document. The *location* represents the location of the file, for example an URL, the *protocol* gives the required protocol to access the file, for an URL it could be *HTTP* or *HTTPS*. The *id* contains an unique identifier for the document. The attribute *refProvider* describes, who deposit the document. Also the segment *interface* repesented by the class with the same name contains through the class *ResourceType* the type of the document. This is in upper letter the type of the document. This is for PDF document the shurtcut 'PDF'. The other possibility to describe an interface is the class *SpecVersion*. Here a Web Service can be defined by WSDL (Web Service Description Language). This is also part of the SAL* interface specification. Therefore the classes *InterfaceDeclr*, *InterfaceSpecification*, *InterfaceOperation* and *InterfaceProperties* are needed. The class *InterfaceSpecification* contains the *id* of the specification and in the segment *operations* are deposit all the operations of the WSDL part.
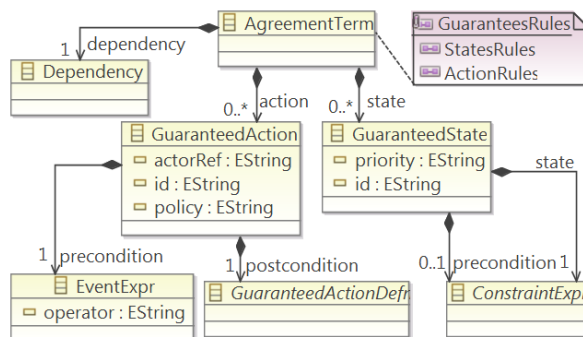
Figure 510: Overview of Agreement Terms

5.1.3.11 Aggreement Terms

The *AgreementTerm* segment is the core part of the SLA* Agreement and is a segment of *SLATemplate*, because it contains the states *GuaranteedStates* and the resulting actions *Guaranteed-Action*. A state describes the relation between a SLO and a measure point based on a interface. In general states contains an attribute *priority*. Both (states and actions) have conditions. The actions must have a *precondition* and a *postcondition*, a state can have many *preconditions*. The representation of conditions happens by the ground expressions of the SLA* model. The actions have an attribute *actorRef*. This field contains the unit, who is responsible to work on it. The class *AgreementTerm* is contained in the segment *term* of the top level SLAs or SLO-As. This is a difference between the SLA* model and the SLO-A Format. Important is, that the class *AgreementTerm* can have only a segment *GuaranteedStates*, when the *type* is not *top-level*. Therefore are exists three state rules:

- A Top-Level SLA (Template) does not hold any states

- An SLO-A (Template) can hold one or more states

- An SLO-A (Template) without a state is a grouping KPI

For *ActionRules* it is essential that a Top-Level SLA (Template) or a SLO-A (Template) can hold any number of actions.

5.1.3.12 Dependencies

The Dependencies of the documents itself is solved simple with the classes *Dependency* and *Property*. Also there are no existing solution in the SLA* model or SLA(T). The idea behind this is, to set the documents itself in relation. All Agreement Term can have many superordinate (segment *depending*) and subordinate (segment *anteceding*) Terms over the class *Dependency* as segment *dependency*. To store the information below the class *Dependency* the class *Property* is used. The attribute *key* contains the name and the attribute *value* references the document either as URI or as UUID. Also rules can be applied. Important is the*CustomerSLO-ARule*. This allows for non customer SLO-A (see attribute *type*) only exactly one entry for *anteceding*. This structure is quiet simpler than the mechanism in the SLA* model.

5.1.3.13 Service Level Objectives

The description of Service Level Objectives contains two components. The first component is the declaration of objectives, its value and the threshold. The threshold will be defined by the variable declaration. This is a mechanism to define a *name*, a *value* and a *data type*. Also the threshold values will be added, that all required information are available. The state definition creates a connection between the in the SLO defined value and at the runtime estimated value

(measure point). The second component of the SLAs is the state definition. This is done by the class *GuaranteedState*. It has a *Priority*, this can be optional set. This class connects the values defined in the SLO and the measured real time values. These information are called measure point. Important is that the name of a defined interface and a UUID of a top level template are required. This relation is known by the SAL* model but was extended. An association to an interface of a SLO-A is possible by storing an UUID of the top level SLA.
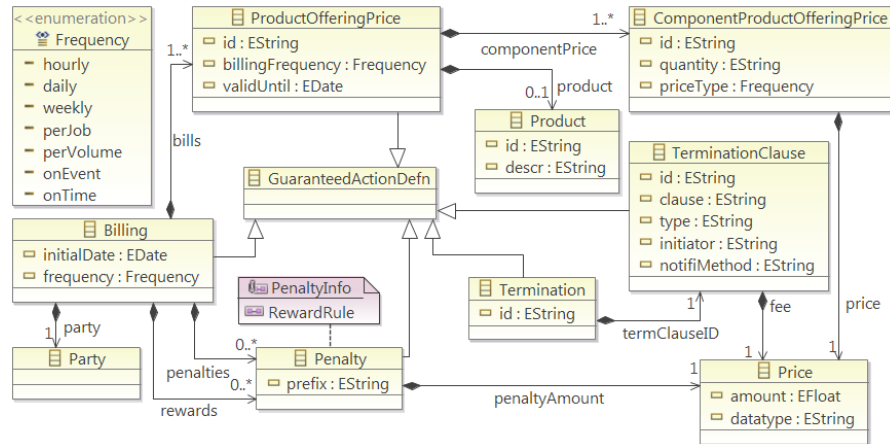
Figure 511: Pricing Business Term Classes

5.1.3.14   Action Definitions

The next part are the actions of the SLO-As. In general there are two different types. *Business Level Guaranteed Actions* and *Pricing Business Terms*. The first one is from the point of system functionality view and is used for the service provision. In this terms and templates are stored information that describes the execution of the system functionality. When an event is happened this information describes all further steps, which must be done. For example this could be an agreement between provider and customer about the start time of the monitoring. Also an example is after an incident, the automatically sending of messages.

5.1.3.15   Pricing Business Terms

The *Pricing Business Terms* describes actions for the agreements and their conditions. Also actions triggers system functions, but they have influence to the service itself. Here are happens actions to costs, penalties, bonuses and termination clauses of the SLAs. A bonus and a penalty are both of the type *Penalty*, but the relation class *Billing* is either *rewards* or *penalties*. The class *Penalty* has an attribute *prefix*, which describes if it is a bonus or a penalty. Also the class *Price* was added. This contains an absolute amount. Also it depends on the relation, if it represents a Price or a free.
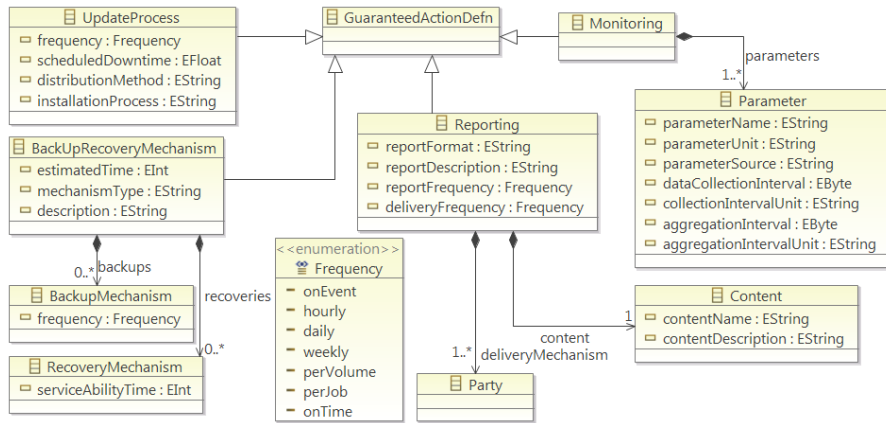
Figure 512: Business Level Guaranteed Action Classes

5.1.3.16 Business Level Guaranteed Actions

The *Business Level Guaranteed Actions* has currently monitoring and reporting action implemented. The monitoring classes are expanded for the SLO-As. Also the attribute *ParameterSource* was added. This allows it, to recognize properties through their names, data types and its source (UUID). Also frequencies for the definition of report, aggregation and acquisition intervals. the following frequency types are added:

- *onEvent* instead of an interval an event triggers the action

- *onTime* a constant time stamp triggers the action

- *perJob* the interval is defined by the runtime of a job

- *perVolume* a concrete value trigger the action. For example 30GB disk space is full, or 1000 connections are open

A major change is the relation to the class *Parameter*. This was changed to composition with *1:n* relation, where *n* must be at least 1. This allows the declaration of multiple parameters and together with the source of the parameter (*SourceParameter*) it is possible to assign values of the depended SLO-As.

5.1.3.17 Use Cases

To illustrate possible applications this section presents use cases and gives the corresponding sample implementations. Cloud computing hugely benefits of the self-serve principle, which underlies it. Therefore it is particular important to give customers the possibility to fill in the SLO templates with only a few steps and in an self-explanatory way. Three use cases will show the modelling possibilities of SLA-A:

5.1.3.18 Availability Use Case

For the first use case we assume that a customer runs a cloud application on an specific regular basis. Therefore he wants to ensure that the resources are available for this specific times and can be used. We assume that the customer already has completed the initial process of creating a valid SLA with an provider for his service. The contract thereby covers the specific service times, in this case every friday from 14:00 to 18:00. This time-frame is therefore stored in the class *ServiceTime* as a *serviceTimePair*, where the key *startTime* has the value 14:00:00 and the key *endTime* has the value 18:00:00. Additionally another field is filled in where the *interval* is set to *weekly* and *friday* is the value for the day so that all the defined agreements, for example the availability or response time are legaly binding for this specific period.

Due to the changing business environment the customer now wants to change the SLA, because he needs the same service to be available on every Wednesday from 09:00 to 14:00. Therefore the customer loads the existing SLA and adapts the *ServiceTime* so that the new requirements are met. The respecting changes to the XML files can be seen in figure 513. In the real life the customer would change these settings by loading the already defined SLA into a tool with an graphical user interface like presented before and would simply change the the *ServiceTime* and generate the new SLA. This however requires that the newly chosen parameters are inside the limitations given by the provider. This validation is done automatic within the adjustment tool. If the validation succeeds the SLA gets signed online by both parties and is legally binding. In the case of mismatching customer demands and provider offerings the SLA can not be signed and a manual negotiation process has to be performed.

5.1.3.19 Additional KPI Use Case

As exemplary second use case a customer wants to extend an existing SLA in order to match the advanced or varied requirements of his services, so the service quality can be better monitored and ensured. In this case the customer wants to add another KPI, which represents his needs the best, to the existing SLA. For the SLA this means that another it AgreementTerm has to be added. Inside this it AgreementTerm the it GuaranteedState, itGuaranteedAction, etc. have to be specified. Additionally the it ServiceLevelObjective has to be created in order to measure and monitor the newly created KPI. If we assume that the customer wants to add the guarantee for the minimum bandwidth his service can use, this KPI and the corresponding it interfaceDeclr can be easily added by using the graphical interface like shown in figure reffig:gui. Therefore the customer chooses a representing KPI from an list of available KPIs, fills in the appropriate values and so adds the new contract clause to the existing SLA. Again it has to be checked if newly added KPI is within the providers range of offerings or not and based on that can be signed online.

```
«DEFINE main FOR ServiceTime»
<sloa:ServiceTime>
      <sloa:ServiceTimePairs>
      «EXPAND xProperty FOR serviceTimePairs»
      </sloa:ServiceTimePairs>
</sloa:ServiceTime>
«ENDDEFINE»

«DEFINE xPoperty FOR Property»
«FOREACH entries AS e»
      <sloa:Entry>
              <sloa:Key>startTime</sloa:Key>
              <sloa:Value>14:00:00</sloa:Value>
      </sloa:Entry>
      <sloa:Entry>
              <sloa:Key>endTime</sloa:Key>
              <sloa:Value>18:00:00</sloa:Value>
      </sloa:Entry>
      <sloa:Entry>
              <sloa:Key>interval</sloa:Key>
              <sloa:Value>weekly</sloa:Value>
      </sloa:Entry>
      <sloa:Entry>
              <sloa:Key>day</sloa:Key>
              <sloa:Value>friday</sloa:Value>
      </sloa:Entry>
«ENDFOREACH»
«ENDDEFINE»
```

```
«DEFINE main FOR ServiceTime»
<sloa:ServiceTime>
      <sloa:ServiceTimePairs>
      «EXPAND xProperty FOR serviceTimePairs»
      </sloa:ServiceTimePairs>
</sloa:ServiceTime>
«ENDDEFINE»

«DEFINE xPoperty FOR Property»
«FOREACH entries AS e»
      <sloa:Entry>
              <sloa:Key>startTime</sloa:Key>
              <sloa:Value>09:00:00</sloa:Value>
      </sloa:Entry>
      <sloa:Entry>
              <sloa:Key>endTime</sloa:Key>
              <sloa:Value>14:00:00</sloa:Value>
      </sloa:Entry>
      <sloa:Entry>
              <sloa:Key>interval</sloa:Key>
              <sloa:Value>weekly</sloa:Value>
      </sloa:Entry>
      <sloa:Entry>
              <sloa:Key>day</sloa:Key>
              <sloa:Value>wednesday</sloa:Value>
      </sloa:Entry>
«ENDFOREACH»
«ENDDEFINE»
```

Figure 513: serviceTime adjustment by the customer

5.1.3.20   Additional Resource Use Case

Another use case for using dynamic SLAs is to extend resources. For example, a customer wants to expand his computing power because of large and short term calculations. In this case the customer can order the new CPU power within the graphical interface, and he also is able to do that with time limitations. But it is necessary for the cloud provider to check the resources before he gives a garantuee to the customer for availability and pricing.

For the pricing it is possible for the provider to give certain offers in regard of predefined

```
<sloa:ProductOfferingPrice>
      <sloa:ID>DefaultOfferingPrice</sloa:ID>
      <sloa:Product>
            <sloa:ID>CPU-Price</sloa:ID>
            <sloa:Description>VM performance per CPU
                  </sloa:Description>
      </sloa:Product>
      <sloa:BillingFrequency>daily</sloa:BillingFrequency>
      <sloa:ComponentProductOfferingPrice>
            <sloa:ID>Dual Core</sloa:ID>
            <sloa:Price>
                  <sloa:Type>per_vm</sloa:Type>
                  <sloa:Value>1</sloa:Value>
                  <sloa:Datatype>unit#Euro</sloa:Datatype>
            </sloa:Price>
            <sloa:Quantity>
                  <sloa:Value>1</sloa:Value>
            </sloa:Quantity>
      </sloa:ComponentProductOfferingPrice>
      <sloa:ComponentProductOfferingPrice>
            <sloa:ID>Quad Core</sloa:ID>
            <sloa:Price> ... </sloa:Price>
            <sloa:Quantity>...</sloa:Quantity>
      </sloa:ComponentProductOfferingPrice>
</sloa:ProductOfferingPrice>
```

Figure 514: Service Offering for CPUs

bundles. In this way the provider is able to achieve a better resource allocation and usage prediction. A sample offering regarding the CPUs can be seen in figure reffig:offering. There a special bundle price for dual core and quad core CPUs is given. The price is calculated by acumullating bundles. In special cases the customer can negotiate new terms with the provider. For the resource allocation the requested demand of the customer is checked against the real time resources of the provider. This is done automatically by inside the SLA creation tool. So after automatically checking the requested resources and calculating the new price the adjusted SLA is available for the customer and can be signed.

5.2 QoS Management Modules

5.2.1 Threshold Value System

Fuzzy Inference

Most approaches consider infrastructure sensor data like bandwidth, request/response time, CPU usage, memory usage, etc. to control the scaling infrastructure as depicted in Fig. 516 (dashed box). The approach of this paper is to use additional, often imprecise information (e.g. weather) to improve the management to meet the QoS requirements stated in SLAs. These imprecise factors (e.g. user wants scaling aggressive/moderate, etc.), political factors (legal changes, political summits, etc.), economic/market factors (product advertising, product launch, etc.), other factors influencing the service usage (e.g. weather, gossip, etc.) can not be modelled precisely.

Fuzzy logic allows to model imprecise information by the user in the form of non-numeric linguistic variables (e.g. weather: bad/good/excellent). These fuzzy inputs are used in the fuzzy control system, that uses expert knowledge to inference a fuzzy output. After defuzzifying this
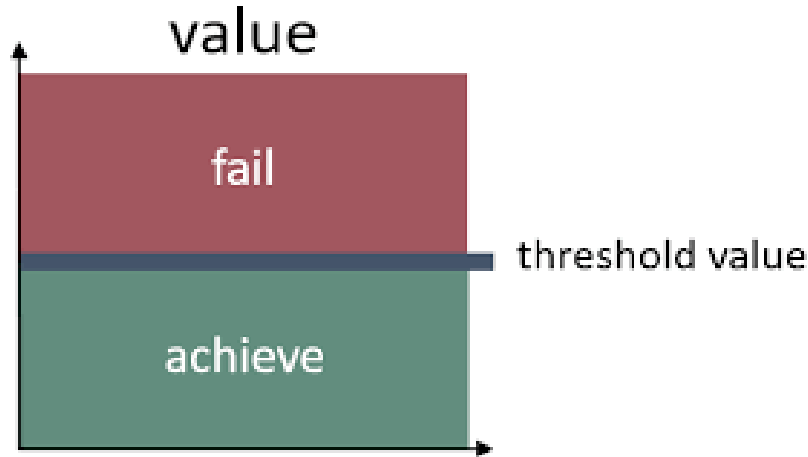
Figure 515:

output to a crisp value, then this states how big the up and down scale factor should be. For example, if a customer wants to have an aggressive scaling control the infrastructure will scale up with e.g. 3 VMs otherwise with only one VM at a time. The scaling domain expertise is modelled in a knowledge base with fuzzy IF-THEN rules.
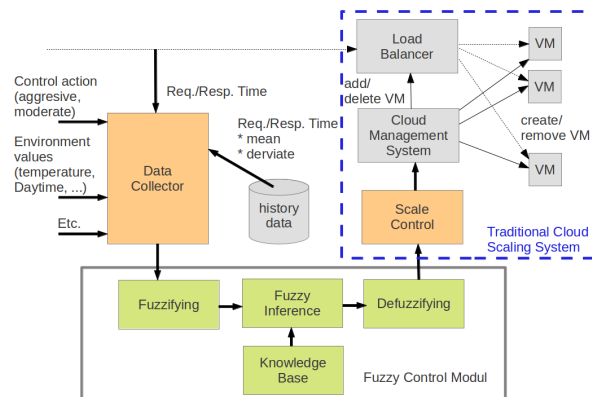
Figure 516: Fuzzy Controlled Scaling Architecture

5.2.1.1 Fuzzy Controled Scaling Architecture

Figure 516 shows the architecture for a load balanced service by automatically scaling up/down the infrastructure by provisioning/deprovisioning VMs. It consists of two new modules compared to the traditional scaling infrastructure (blue box), the *Data Collector* and the *Fuzzy Control Module*.

The *Data Collector* collects all information data, crisp (e.g. cpu usage) and imprecise data (e.g. weather). The data is categorized in infrastructure data (e.g. req./resp. time), history data (e.g. req./resp. time 5 minutes ago), control action (e.g. aggressive up/down scale), environment data (e.g. daytime), and other information that might influence the load of the service. Some of the data is automatically monitored, like the infrastructure data, some is computed like the history data, and some is determined by users/experts like control action or environment data. All collected data is input data to the *Fuzzy Control Module*, where the data is fuzzified, results are propagated by the fuzzy inference engine and quantified by defuzzification. The defuzzified value (Number of VM to be started or stopped) is put into the *Scale Control* module, which generates XML-RPC calls to the *Cloud Management System*.

5.2.1.2 Collected Information Factors for the Fuzzy Control

The relevant information to improve elasticity can be categorized into the following monitoring data: infrastructure, infrastructure history, time-dependent, and service-dependent sensor data.

Infrastructure sensor data includes factors that can mostly be monitored using sensors placed in various locations and layers of the cloud infrastructure. KPIs, like request response time, which can easily be measured at the load balancer (LB). Cloud specific parameters, like start time of VMs, can be aquisitioned at the cloud management system. The quality of the cloud infrastructure or service implementation can be taken into account as well. The load balancing control might be influenced by the basic robustness of the overall infrastructure. The infrastructure robustness can be modelled by an imprecise parameter e.g. strong, weak.

Infrastructure history sensor data are parameters that have been previously collected into a history data base. The purpose is to calculate values like, mean value, derivation value, etc. These statistical data can be good indicators to improve the LB management. Imprecise history parameters can be of interest as well. Suppose a service depends on the weather condition (e.g. online shop for winter tires), then a sudden change of the weather condition from dry to snowy condition makes it more likely, that the load of such a service is higher.

Time-dependent sensor data contain parameters that can influence the infrastructure management at a predefined time. The knowledge of the typical weekly usage for an service (see Fig. 517) can be modelled and therefore the decision to scale up or down strongly or weakly depending whether the change is high or not.
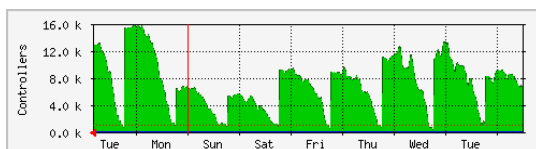

Figure 517: Example: Weekly Load of the HFU Learning Management Platform

Service-dependent sensor data involve parameters that influence the control infrastructure

depending on the related service. Political parameters, like new legal issues enforcing more logging at the service side. Market events, like product launches, marketing events, new prices, etc. can influence the usage of services. Gossip, modeled as good news or bad news is influencing service usages. Importance of service might need a more aggressive management to make sure, that the SLA violations can be minimized.

5.2.1.3 Fuzzy Control Module

The *Fuzzy Control Module* consists of four main fuzzy control processes represented by the four sub-modules respectively (see Fig. 516). The crisp and imprecise input data is converted into fuzzy values for each input fuzzy set with the *Fuzzifying* module. The decision making logic of the *Fuzzy Inference* module determines how the fuzzy logic operations are performed (SUP-MIN inference), and together with the *Knowledge Base* module determine the outputs of each fuzzy IF-THEN rules. Those are combined and converted to crisp values with the *Defuzzification* module. The output crisp value can be calculated by the center of gravity or the weighted average and are converted to the number of VM to started or stopped.
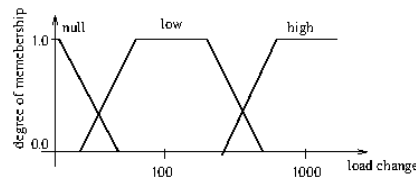
Figure 518: Input Fuzzy Set for Load Deviation

Fuzzification Fuzzification is the process of decomposing the input data into fuzzy sets, with trapezoidal shaped membership functions. Figure 518 shows a system of fuzzy sets for an input with trapezoidal membership functions. Any particular input is interpreted from this fuzzy set and a degree of membership is interpreted. If the request-response-time, for example, it set to about 100 request-per-seconds, the fuzzy value $loaddeviation$ is set to low.

Fuzzy Inference The fuzzy values gathered from the input data are processed by the inference engine using the expert domain knowledge modeled as fuzzy IF-THEN rules. The following fuzzy rules are examples how to state the domain knowledge in the area of up and down scale control.

```
IF ReqRespTime_rising=high AND
   expected_ReqRespTime_rising=high AND
   product_launch=now AND
   ....
THEN
```

57

```
    up_scale=very high
...
```

Defuzzification   After the fuzzy reasoning the resulting linguistic output variable (e.g. scale up = high) needs to be translated into a crisp value (e.g. number of VMs to be started or stopped at time). Defuzzification maps the output from the fuzzy domain back into the crisp domain. The most common defuzzification methods is the Center-of-Area (C-o-A) often referred to as Center-of-Gravity and is defined as follows:

$$(5.1) \qquad x^* = \frac{\int \mu_i(x) x \, dx}{\int \mu_i(x) \, dx}$$

where x^* is the defuzzified output, $\mu_i(x)$ is the aggregated membership function and x is the output variable. The C-o-A method calculates the area under the scaled membership functions and within the range of the output variable and afterwards calculates the geometric center of this area.

5.2.1.4   Simulation Environment

In order to test the feasibility of our approach, we created a simulation environment to be capable of validating the general fuzzy controlled scaling architecture proposed in this paper. The simulator therefore consists out of four major modules (see Fig. 519). Firstly, the *request generator module*, which simulates the real life usage by generating requests to the cloud service. It is important to note, that in this simulation requests are generated with an static predefined workload trend. The second component is the *load balancer module*, which distributes the generated requests equally to the pooled Virtual Machines (VMs) by round-robin. Here also the request response time is determined, therefore the time from generating the request until the response arrives at load balancer is measured. Within the *logic modules* this determined request response times then are checked and based on the stored fuzzy or conventional rules will be decided to scale the service up, down or do nothing. And finally the *scaler* , which connects to the actual cloud management system, which is responsible for the provision and adding or removal of resources. The used conventional rules are represented by an simple boundary system based with a low and high threshold. When the measured average request response hits the upper boundary a VM gets started or when the lower boundary is hit a VM is stopped. These boundary systems are widely used and provide a common way of QoS management. The fuzzy set basically uses the same boundaries as the conventional rules, but as additional decision factors, an prediction based on expert knowledge or outlooks is used. This additional information allows an earlier respond to the changing loads and thus an better reaction to upcoming demands. Based on historical data the approximate usage and thus the load can be determined for a service. Figure 517 show the historic data of the load for an service during a week, based on this a simplified load prediction during daytime can be derived.
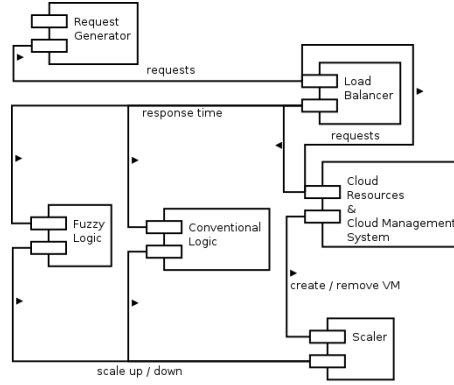
Figure 519: Simulator Module Diagram

Based on such knowledge an expert specifies whether the load will expectedly be increasing at an high, regular or low rate. An predicted regular load will result in an rule set that equals the conventional boundaries, therefore resulting in essentially the same behaviour. In case of an high prediction, the scaler generally scales up faster, which means VMs are started on a lower load and additionally up to two VMs can be started at the same time based on the load. Simultaneously the scale down boundary is set to a lower load to keep a higher pool of available VMs. The low prediction, is in principle a reversed high prediction, which will change the behaviour into generally scale up later and scale down faster and up to two machines at once.

As an additional basis for decision, the slope of the last measured points is used. The basis for this assumption is that a strong growth of response time indicates a upcoming peak load. The slope factor is therefore calculated of two response time values with the following formula:

<div style="display:flex; align-items:center;">

(5.2)

$$slope_a = \frac{\Delta Y}{\Delta X} = \frac{Y_2 - Y_1}{X_2 - X_1}$$

</div>

Where Y and X are the time and value of the measured response time and $slope_a$ is the currently determined slope. Since the course response time is unlikely linear, the slope must be determined more than once in order to reconstruct the actual system behaviour. Problematic are load bursts, which the determined slope suddenly rises or falls down extremely. In the worst case this could cause a VM to be started up and immediately get shut down again or vice versa. Therefore, the average of of the last n slopes are used for the assessment. The simulator is based on a model in which a generated request includes a static processing time of 100ms. The KPI, is measured as the request/response time, based on the average of the last 10 processed requests. Thereby the time is counted form the generation of the request, till arrival of the response after the processing at the load balancer. The QoS limit has been set to 2000ms in this model and the conventional rule set regulates at an average response time of 1500ms by up-scaling and at

1000ms by down scaling one VM at a time. To eliminate the influences of the test environment, like processor fluctuations the factor of 10 was used to all above described values.

To determine the suitability of the procedure presented, different scenarios have been created and tested with and without the fuzzy control mechanism.

5.2.1.5 Simulation Scenario

In the presented scenario an complex course of load is sent to the service. In the first three minutes there is an peak demand, followed by reoccurring burst loads. This scenario simulates an peak load that doesn't flatten quickly. This could be the case for services that depend heavily on news (e.g. launch of intensive product advertising). If the related service shows up in the news a initial large peak is produced and is recreated as smaller recurring peaks by spreading the word. Figure 520 shows load graph for this scenario.
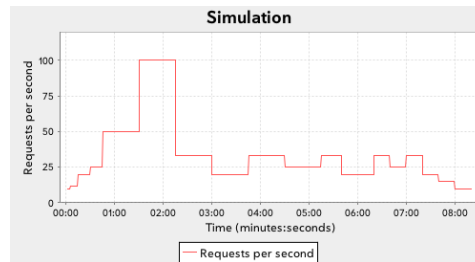
Figure 520: Load Graph

Result with conventional rule set: The conventional rule set result graph shows that in this test the QoS limit of 20,000ms is exceeded for the first initial peak load and is nearly hit for the following rises. The scaling lags behind compared to the load increase. So it can not cope with the increasing load fast enough by starting new VMs. This is clearly proven by the fact that the
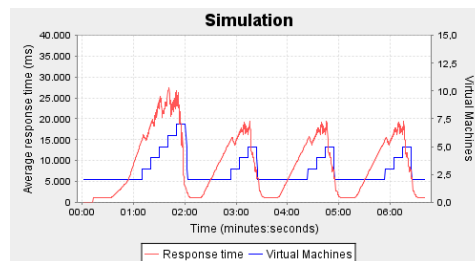
Figure 521: Conventional Rule Set Results

maximum number of VMs simultaneously used is only reached immediately before the flattening of the load.

Result with fuzzy rule set (high prediction): Compared to output of the fuzzy rules with *high* prediction, as shown in figure 522, it becomes clear that here the VMs are much earlier for disposal allowing to reduce the response time and be inside the SLA margin. Although for the first peak load two VMs additional VMs are used, this resembles a relatively small expense in compare to breaking the SLA. It is shown later on in the simulation that better results are delivered in the smaller peaks with the same number of VMs.
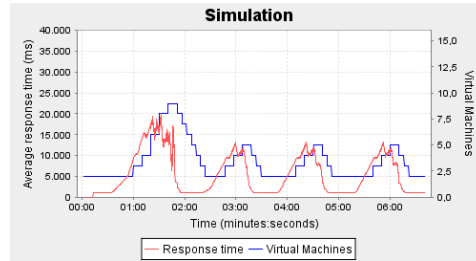
Figure 522: Fuzzy & "High" Prediction Results

Result with fuzzy rule set (high prediction & slope): Comparing these results with those of the fuzzy rules with *high* prediction and *slope* (see Figure 523), the response time has been reduced again, in spite of the very rapid scale up of VMs. The response time is in the range of all other peaks and is in this case is maintained very spacious. However, in this case an additional VM is switched on compared to the fuzzy rules with *high* prediction. The large differences in the results graphs are however not dependent on this additional resource. Instead the up-scaling of several VMs simultaneously achieves the improvement. In this case it is up to 3 VMs are allocated at the same time.
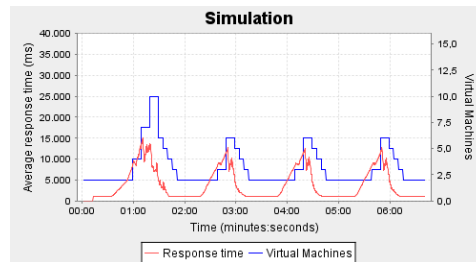
Figure 523: Fuzzy & "High" Prediction & Slope Results

It can be concluded that with the presented approach it is clear that more resources are used which can lead to higher costs for the customer. But the response time and the SLA thresholds are maintained better and less SLA violations occur. Whether this is economically in each case must be decided by experts contemplating the SLA and contract data.

5.2.2 Machine Learning Algorithms

To improve the output of the QoS Management Module, different machine learning algorithms were used to test the feasibility of the approach. Firstly a general Artificial Neural Network was used to predict different environmental KPIs. Afterwards an investigation and comparison of Support Vector Machines, Artificial Neural Network, and general Linear Regression was conducted.

5.2.2.1 Artificial Neural Network

«««< HEAD The aim of this first approach was to create a prototype application which enables efficient provisioning of cloud storage resources with the use of Artificial Neural Networks to achieve better compliance with SLAs. The most common type of ANNs used for forecasting is the feed-forward multilayer perceptron (ffMLP), as seen in Figure 524. ======= Artificial neural networks, in short: ANN, are networks of artificial neurons which aims to simulate the process inside the human brain and represent a branch of artificial intelligence. Artificial neural networks have been successfully used in the past with all sorts of optimization problems, which makes them a suitabel approach for our solution. The aim of this first approach was to create a prototype application which enables efficient provisioning of cloud storage resources with the use of Artificial Neural Networks to achieve better compliance with SLAs. The most common type of ANNs used for forecasting is the feedforward multilayer perceptron (ffMLP), as seen in Figure 524. »»»> f2b6e9925dd900f7fe283a6949be98ab35e3fd0f
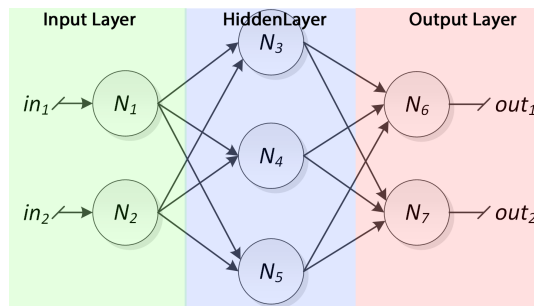
Figure 524: Simple 3-tier Feedforward Multilayer Perceptron.

These are Neural Networks, which consist of one input layer, n-hidden processing layers and one output layer. Feed-forward networks are classified by each neuron in one layer having only direct connections to the neurons of the next layer, which means they have no feedback. In feed-forward multilayer perceptrons, a neuron is often connected to all neurons of the next layer, which is called completely linked. So, there is no direct or indirect connection path from neuron N_x which leads back to a neuron N_{x-z}. To compute a one-step-ahead forecast, these NNs are using lagged observations inputs of time series or other explanatory variables.

For the creation of the Neural Network model we used the graphical editor and simulator MemBrain [**?**]. The presented Neural Network consists of 119 neurons, which are aligned into 5 layers, and corresponds to a ffMLP where not all neurons are completely linked. An architectural overview of the presented model is shown in Figure 525 below.
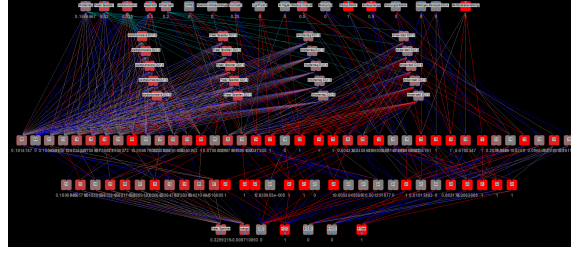
Figure 525: Feedforward Multilayer Perceptron Architecture.

Training of ANNs can be seen as a complex non-linear optimization problem, and sometimes the network can get trapped into a local minimum. ANNs can theoretically learn by developing new or deleting existing connections, changing the connection weights or threshold values, altering one or more of the three neuron functions (activation, propagation and output) and developing new or deleting existing neurons. In order to improve outputs, the input neurons should get normalized variables. This can simply be done by the equation below.

(5.3)
$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

In order to avoid local minima and bad results, the training should be initialized several times with different starting weights and alignments. For the training of the proposed model, data sets were created in the form of Comma Separated Value (CSV) files. Each file contains storage usage patters with input and output vectors. Here, 60% of the samples were used for training and the remaining 40% were used for the validation of the network. The output behaviour was modelled by depending on a input vector, where the desired output values where manually entered into the input vector. Thus, situations in which the responsible output neuron shall increase the amount of allocated memory were mapped.

To teach the network the prediction capability of future memory usage, the input vector was extended. The entire course of the used memory amount was added for the period of t_0 to t_n. The desired output for this input vector at the given time t_i shall be the predicted amount of memory used at time t_{i+x}. To achieve this, the value of the output vector at any point t_i in the time period t_0 to t_n was set to the input vector of the point t_{i+x}, by which x determines the length of the forecast period. Through this shift in values ‚Äã‚Äãthe network can be trained for a prognosis. During each training session the network error was checked with validation data. MemBrain

calculates this using the following formula:

$$(5.4) \qquad NetError = \frac{\sum\limits_{i=1}^{n} (Target_i - Output_i)^2}{n}$$

The desired activation of the output neurons is here referred to as $Target$ and the actual calculated activation is the $Output$. The squared deviations are summed and divided by the number of training data sets. To determine whether the Neural Network shows good results of the output behaviour, it has been trained and validated with 10 different training data sets. The result for the network error after each learning processes is shown in Table 51 below.

Table 51: INFRASTRUCTURE SENSOR PARAMETER.

TrainingNr.	NetError(Training)	NetError(Validation)
1	0,0000573	0,046
2	0,0000586	0,040
3	0,0000713	0,040
4	0,0000702	0,112
5	0,0000611	0,040
6	0,0000783	0,083
7	0,0000703	0,046
8	0,0000627	0,038
9	0,0000645	0,061
10	0,0000630	0,046

Here, it can be seen that the NetError reaches overall good values close to zero and not only for a particular dataset. The average total error for all training runs from Table 51 is 0.0000657 for trained and 0.0573 for untrained (unknown) input data.

Evaluation The aim of this prototype was to investigate, whether or not the use of a Artificial Neural Network for the provisioning of a cloud storage resources has a positive effect on SLAs compliance, and whether this can lead to a better resource utilization compared to a classic threshold value system. For this purpose we created a simulation environment where storage requests (read, write, and delete) form a generator were sent trough a QoS monitor. Inside the QoS control module, the Artificial Neural Network and the threshold value system were used to regulate the amount of allocated storage capacity. Figure 526 shows the architectural overview of the simulation environment.

In the simulation, the impact of regulatory mechanisms on the following key performance indicators was considered:

- Free memory amount: providing an optimal amount of memory by the control logic.

- Response time: compliance with the KPI response time by adjusting the storage medium.
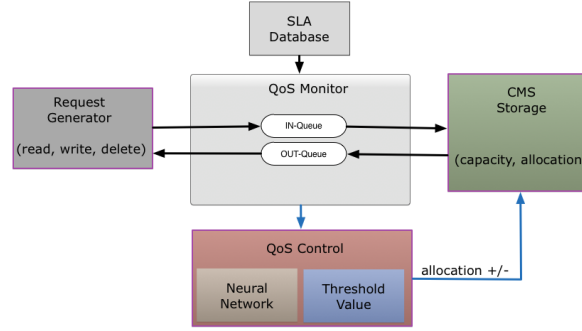
Figure 526: Simulation Architecture.

- Backup Media: proposal of a suitable backup medium.

For this, the used Neural Network consisted of 11 different input neurons. Table 52 lists the used input neurons and describes the used input factors. As output neurons, there is one neuron that gives the expected used memory amount for the next simulation step, a neuron that determines the amount of memory to be added or removed, as well as other neurons that recommend the optimal backup medium.

Table 52: SIMULATION INPUT NEURONS

Neuron	Description
Time	Point t_i in $t_0...t_n$
Weekday	Day of week for point t_i
Free Storage Capacity	Free storage capacity at point t_i
Growth Rate	Change of capacity from t_{i-1} to t_i
Response time	Mean response of last 5 inputs $\frac{\sum_{i=i-5}^{i} t_i}{n}$
Queue Length	Still open request at point t_i
Troughput	Troughput at point t_i
Access Rate	Amount of requests per time slot
Request Type	Distinction between large and small requests
Backup Amount	Size of backup data
Bandwidth	Usable bandwidth at point t_i

In order to compare the results of the Neural Network with a common, in practice widely used method, a threshold value based scaling was implemented. This regulation system is controlled by predefined thresholds for the monitored KPI values. The implementation for the threshold rules for adding and removing allocated storage can be seen below in Figure 527, as simple pseudo-code if then rules.

```
// addStorage
IF AllocatedCapacity       UsedCapacity < SLAFreeCapacity +2
THEN
        IF (UsedCapacity < 20)     AllocatedCapacity += 10;
        ELSE IF (UsedCapacity > 80)    AllocatedCapacity += 20;
        ELSE   AllocatedCapacity += 15;

// removeStorage
IF AllocatedCapacity       UsedCapacity > SLAFreeCapacity +15
THEN
        IF (UsedCapacity < 20)     AllocatedCapacity -= 20;
        ELSE IF (UsedCapacity > 80)    AllocatedCapacity -= 10;
        ELSE   AllocatedCapacity -= 15;
```

Figure 527: IF THEN rules for threshold system.

Here, it can be seen that, by falling below a 2% buffer of the storage value defined in the SLA, the allocation will be increased and by exceeding 15% over the amount of storage defined in the SLA, the allocation will be lowered. The amount of which the allocated storage will be changed is dependent on how much the overall storage usage is. In case of an usage of over 80 %) increase will be 20%, with an usage of below 20% the increase will be 10% and in between the increase is 15 % of the overall volume. These settings are reversed for the deallocation of the storage.

For the scenario in this simulation, a dynamic storage SLA, in which a customer gets granted 10GB of free space and up to 100GB of overall usage, was assumed. With such a dynamic limit described in the SLA, it is particularly important for the provider to find a solution that is as close as possible to the guaranteed amount of storage, since this will ensure a high economic efficiency. In practice, however, this usually is not possible. For this reason and because a violation of the SLAs can have monetary consequences, bigger buffer zones are installed. Figure 528 shows the resulting graph of the simulation with the conventional threshold value rules.

Figure 528: Storage allocation results for threshold rules.

The red graph in Figure 5 and Figure 6 shows the course of the memory usage in GB, by the user during the simulation. The usage has been pre-generated for the simulation purpose and shall resemble a system, where a user regularly creates and deletes files with up to 15GB size, as well as generate larger files with up to 50GB. This type of usage may occur while working with different media files, like in the post-processing of movie projects. The green line marks the guaranteed amount of storage available to the user, granted by the SLA. It proceeds synchronous to the red graph, since the user gets guaranteed 10GB more than they currently use. The blue

graph shows the pre-allocated amount of storage, which is directly usable by the user.

If we compare these results with those obtained by the Neural Network controlled storage allocation, shown in Figure 529, it becomes clear that the efficiency is marginally improved. With an average of 18.68% of memory over provisioned the threshold value system is almost as effective as the neural network, with a 18.22% overhead. The slight difference arises from the fact that the allocation offered by constantly adopting, fits to the SLA limits with a relatively constant overhead. In contrast, the threshold value system initially provides too much memory, and then only adopts the amount of allocated storage shortly before a violation of the SLA it to occur.

While comparing the two graphs, we see that the threshold system due to the fixed thresholds less often adjusts the amount of memory (blue curve). Since the added / removed amount of memory operates with a fixed predefined value‚Äã‚Äã, often too much memory is provided and then immediately gets removed again. This happens likewise when reducing the amount of memory allocated, which often leads to falling below the specified minimum amount in the SLA. However, the Neural Network determines constantly, based on the learned training data, a variable amount of memory that is to be added or removed, which leads to adequate reactions and a slightly better economic result.
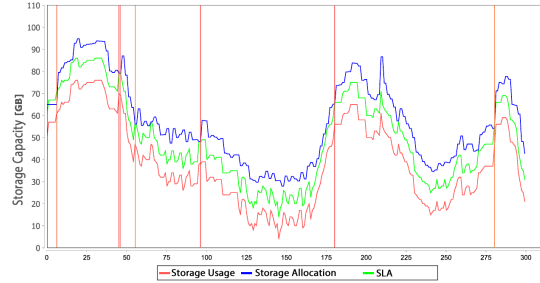
Figure 529: Storage allocation results for NN.

However, when comparing the number of SLA violations, it becomes clear that the Neural Network approach delivers a significantly better solution. This is also evident in the resulted graph seen in Figures 529 and 528, where the SLA violations are indicated by vertical red lines. These exemplary results of the simulation show that the Neural Network produces 7 and the threshold value system 13 SLA violations. These results were also confirmed within the other test runs, where the Neural Network generates an average of 7.45 violations per run and the threshold values system of 15.03 SLA violations per run. Overall, the Neural Network generated solution for the provisioning of storage is better suited, since the number of SLA violations is significantly lower. Together with the slightly lower overhead makes this a reasonably good solution.

5.2.2.2 Support Vector Machines

5.2.2.3 Evaluation

In order to apply and evaluate different Machine Learning algorithms, the open source software RapidMiner [114] was used. The log files created by the CloudSim application were utilized as training and test sets. Furthermore, the available Series extension provided by RapidMiner was used. This extension enables an efficient way to quickly replace different Machine Learning algorithms during the process of creating and evaluating a model in regard to ordered time series.

With the help of a horizon of h=20 (2 seconds), it was defined that the learning algorithms gets to learn the next h time steps in order to be able to predict the value of the average response time of h+1. After the prediction, the time window is incremented by 1 and the next value gets predicted.

- **Neural Network:** Feed forward NN, training back propagation | Hidden layers: 8 | Training cycles: 1000 | Learning rate: 0.3 | Momentum: 0.2 | Decay: true | Normalize: true | Error epsilon: 0.00001

- **Support Vector Machines:** Kernel Type: radial | Kernel Gamma: 1.0 | Kernel cache: 200 | C: 0 | Convergence epsilon: 001 | Max iterations: 10000 | Scale: true | L pos: 1.0 | L neg: 1.0 3) Linear Regression: | Feature selection: Iterative T-Test | Max iterations: 1.0 | Forward alpha: 0.05 | backward alpha: 0.05 | eliminate co-linear features: true | min tolerance: 0.05 | use bias: true

- **Linear Regression:** Feature selection: Iterative T-Test | Max iterations: 1.0 | Forward alpha: 0.05 | backward alpha: 0.05 | eliminate co-linear features: true | min tolerance: 0.05 | use bias: true

Scenario 1   The first testing scenario resembles a classical burst load. Here the server is hit with short bursts of high demand. In out test the requests at the server are highly increased during 10 seconds followed by a phase of low requests. This pattern ist repeated four times during the course of this test. This type of load can occur, for example, on servers that are attacked by botnet (denial of service attack). Alternatively, this pattern can also occur with automated access to APIs or synchronized access to data. In reality, this kind of load is particularly difficult to process since you can adjust it badly and depending on the scale used the system simply can be too slow.

- *Neural Network:* In figure 530 it can be seen, that the NN overestimates the peak of the burst loads in every case. Also it can be seen that the difference between predicted peak and real peak is the biggest during the first burst and that there is an improvement when predicting the later peaks of the bursts. The briefly following decline and rise after each
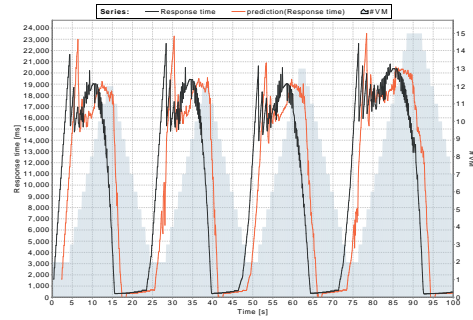
Figure 530: NN Scenario 1: 0s-100s
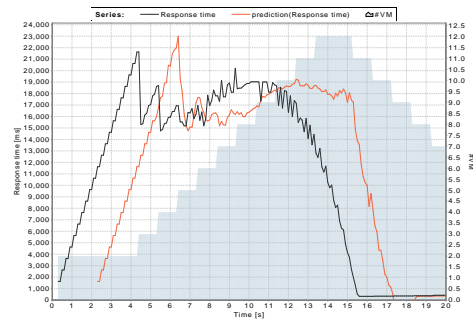
Figure 531: NN Scenario 1: 0s-20ss

peak, e.g., during sec 8-15 is respectively underestimated and overestimated but it can clearly be seen that there is an improvement in the last iteration. Figure 531 shows the delay characteristics and that the algorithm in general can adapt well to the problem.
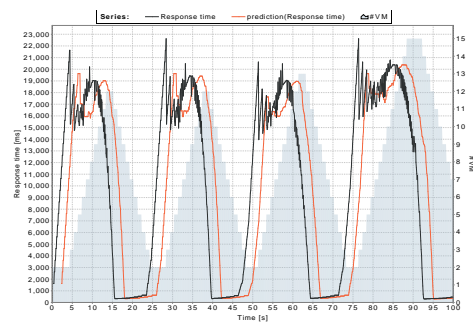
Figure 532: SVM Scenario 1: 0s-100s

- *Support Vector Machines:* Figure 532 shows a contrast to the NN algorithm. In the case of SVMs the first peak of each burst is underestimated. The following cooldown phase before the second peak of each burst is overestimated but an improvement over time can be seen,
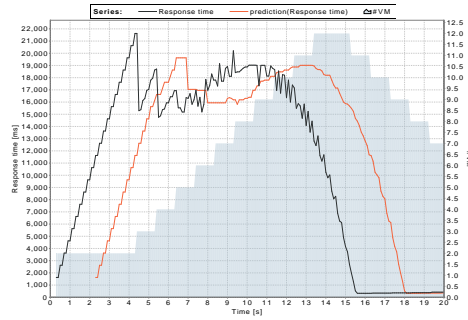
Figure 533: SVM Scenario 1: 0s-20ss

especially on the last burst. Figure 533 looks specifically at the first burst and a comparison to the NN 531 makes it clear that SVMs predict a more smooth curve. It should be kept in mind that it is realistic to assume that in real life there are scenarios with different requirements regarding the reaction to those predictions where this specific differences, smooth or rough, could be seen as either an advantage or disadvantage.
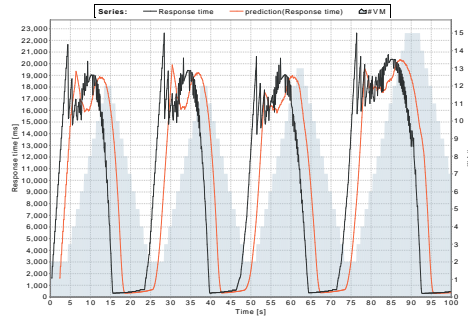
Figure 534: SVM Scenario 1: 0s-100s
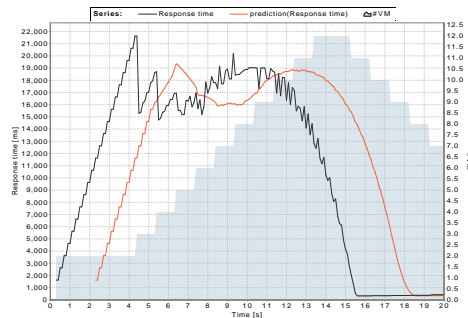
Figure 535: SVM Scenario 1: 0s-20ss

- *Linear Regression:* The predictions made with the help of a Linear Regression learner

shown in figure 534 seem to be very similar to those predictions made by the SVM in figure 532. This insight is further substantiated by taking a closer look at the bursts in figure 535 and figure 533 where a similar prediction pattern can be seen. Worth mentioning is that the predictions made by Linear Regression lead to an even smoother curve compared to the curve predicted by the other 2 algorithms.

Scenario 2 The second scenario shows a system with a normal usage load. The load slowly builds up and then slowly decays again. This scenario in this form or alternatively with strongly decaying load at the is a normal use of the system such as a file transfer or the curl of an larger API. The scenario duration of twenty seconds is in the normal real life range for such actions. This load is easier to scale because the duration and therefore the speed needed to process this is lower.

- *Neural Network:* When looking at the overview in Figure 7 it is demonstrated that moderate changes in response times are learned rather well. The interesting part, shown in more detail in Figure 8, showcases the nature of overestimation. Again, the peak is overestimated, but the predicted curve recovers very fast and yet this issue occurs again after the second plateau. It should be noted that with a different configuration of the NN algorithm a very different curve can be predicted. For this paper we looked at a specific configuration of the algorithm because this characteristic can be utilized and will be explained during the comparison of the algorithms.

- *Support Vector Machines:* The overview shown in Figure9 displays the capability of the algorithm to be able to adapt to a singular, steadily climbing response time. The prediction during the first phase (0-60s) is handled well by the algorithm. Figure 10 illustrates that the spontaneous and large decline in response time is learned very well. This is an important characteristic as predictions based on those quick changes could be the focus during the application in real-time scenarios. None of the other algorithms is able to predict scenario 2 this precisely.

- *Linear Regression:* Figure 11 shows again great similarity between predictions made with the help of Linear Regression and SVMs. Again the difference is that the ascent of the curve is predicted in a smoother way. Additionally, the spontaneous and large decline seen in Figure 12 is not predicted very well. The same characteristic applies on the following smaller decline. As a conclusion it can be said that in this specific scenario the model trained by Linear Regression is the weakest.

Scenario 3 The third and final scenario is a mixture of the two previous ones. First, a single burst is triggered, followed by a slower but higher load, which decreases in the end. Such mixed load profiles are very often found in reality because different users and systems access resources at the same time. These loads are not synchronized and therefore can lead to chaotic scenarios.

- *Neural Network:* Figure 13 shows that during the first phase (0-100s) the model trained by a NN has minor problems in predicting the response time. Although the peaks are generally predicted well, sometimes they are underestimated and sometimes overestimated. But in contrast to the other algorithms the difference in error margin is very small in most cases. During the recovery times after each slope, the local minima are overestimated almost in every case. While the first big peak of a response time over 42000ms is overestimated as well, the second one is predicted almost perfectly. The relative smooth slopes before, during and after the larger peaks are predicted very well with no prominent deficit.

- *Support Vector Machines:* Figure 14 shows that a model trained by SVMs can predict the response time for a varying scenario rather well. The occurring peaks during the first phase (0-100s) are underestimated in every case, but not to a large degree. This leads likewise to the underestimation of the recovery times after each peak, which are the consequence of adding and deleting Virtual Machines. The two larger peaks with a response time of over 42000ms are underestimated again by a small margin while the relative smooth slopes before, in between and after are learned well with non prominent deficit in their prediction.

- *Linear Regression:* Figure15 shows that a model trained by Linear Regression can cope well in a varying scenario. Similar to the SVM model it slightly underestimates the response time in the first phase (0-100s). In general, it can be said that those models are very similar and have only minor, negligible differences. The main difference is that the use of Linear Regression leads to smoother slopes.

While it was shown that all 3 algorithms can be effectively used for predicting the response time in different scenarios it can be said that the NN has a minor advantage over the other algorithms. The main reason is that the NN, in general, slightly overestimates and almost never underestimates the response time. The practical application of this knowledge, e.g., using those predictions in combination with a scaler who manages the quantity of VMs leads to a more assuring state that requirements like defined SLAs can be covered more carefully than with other algorithms. In less critical business cases, where the defined SLAs and response times are not that sensitive, the other 2 algorithms, SVMs and Linear Regression, can be used despite their tendency to slightly underestimate response times. Especially the Linear Regression with its fast training and deployment times could be considered in near real-time scenarios.

5.3 Holistic SLA Management

As described in the previous sections, the quality, respectivly the KPIs of a service can be improved by the described methods. However, it is not enough to concentrate merley on the quality of the individual services, since in modern cloud landscapes, disruptions, failures or performance issues can occur. This requires a holistic approach, where the management of the individual service

quality and the overall management of the cloud in brought together. For example, in a general cloud landscape it is quite possible that the individual services are scaled and managed almost optimally by the algorithms described above, but the overall performance of the landscape is the limiting factor. This is where the holistic SLA management comes to bear. Here, not only on each service as a single is respected but also the overall situation of the landscape is taken care of. For example, in an cloud enviornment where the performance is adversely affected by an infrastructure problem, this may mean that certain services are sacrificed in favor of other services. In particular, the contractual penalties and categorization of the customer regulated by the SLAs play an overriding role.

5.3.1 Instance Sheduling with Genetic Algorithm

5.4 Results and Evaluation

5.4.1 Scenario A

Request Response time SLA

5.4.2 Scenario B

HDD and Backup SLA

5.4.3 Scenario C

SLA Management to Cost / Fine minimization

The related work section presented in this thesis, focuses on the state of the art in Cloud Computing in regard to it's monitoring, scheduling and SLA management capabilities. Therefore this chapter was divided into five categories, namely: (1) Infrastructure Measurements and Cloud Monitoring, (2) SLA Description Languages, (3) Performance Prediction, (4) Scheduling Mechanisms and (5) SLA Enforcement.

6.1 Infrastructure Measurements and Cloud Monitoring

Cloud Computing today delivers nearly unlimited scalable on demand computing resources within a few clicks. But the monitoring and government of such resources may come with some requirements. Monitoring is one of the fundamental parts of every cloud platform, since monitoring is needed to scale applications or instances correctly based on their resource utilization. It is also need to detect and discover defects and limitations, such as bottlenecks in the infrastructure environment. Additionally monitoring can give viable insights to both Cloud users and providers by revealing usage patterns and trends. Without any form of monitoring Cloud providers would not even be able to invoice their customer correctly, since they would not be able to tell how much ressouces and for how log the customer has used them. Cloud Monitoring has many ties into different Cloud Management fields such as capacity and resource planning and management, SLA management, incident management and troubleshooting, performance management and billing. An early example towards distributed resources monitoring is NetLogger [4], a distributed monitoring system, which could monitor and collect network information. Published in 2000, application could use NetLoggers API to gather load information of the network and react accordingly. With GridEye [57] a service-oriented monitoring system, with an flexible, scaleable architecture and forecasting algorithm for performance prediction on the basis of traditional

MA(k) and ExS(alpha) models was purposed. In 2009 Sandpiper [144] was proposed, a system, which automatically could monitor and detect hotspots and based on that remap or reconfigure VMs if necessary. The described system marks a first step towards autonomous SLA management, since the internal remaping algorithm considered SLA violations and to avoid them. In recent years various academic and commercial Cloud monitoring solutions have been proposed. In 2014 Jonathan Stuart Ward and Adam Barker [143] published a taxonomy of Cloud Monitoring stating scalability, cloud awareness, fault tolerance, aromaticity, comprehensiveness, timeliness and multiply granularity as core requirements towards effective Cloud monitoring. Fatema et al. [47] and Aceto et al. [8] analized the most common open source and commercial monitoring solutions, such as Nagios, Collectd, Ganglia, IBM Tivoli, Amazon Cloud Watch, Azure Watch, PCMONS, mOSAIC, CASViD and many more, according on how they relate with these requirements. Recently Manasha Saqib [119] apportioned these requirements along the seven layers of Cloud Computing accoring to [31] [129] [130]: Facility, Network, Hardware, Operating System (OS), Middleware, Application and User. These studies provided a comprehensive overview of the available monitoring tools and their ability to support Cloud operational management, but also stated their shortcomings in the different areas. Cloud Monitoring in this thesis is considered as an enablement technology. The proposed architecture and algorithms are able to work with different monitoring solutions. The main focus thereby lies within the acquisition and enhancement ot such methodologies with additional information. An additional topic especially in the area of PaaS is the application monitoring and application performance monitoring in the cloud.The definition of application performance management by Menasce [100]: APM is a collection of management processes used by organizations to ensure that the QoS of their e-business applications meets the business goals. So APM directly aims at the application lifecycle and management processes such as monitoring, resource management, performance-management, reporting and so on, of software systems. According to Gartner [**?**] there are severall established big names in APM such as IBM, Oracle, BMC and so on, that are getting challenges by innovative start-ups such as AppDynamics or Dynatrace, and Correlsense. APM can be used as performance analysis and monitoring agents in SaaS and PaaS, and as such will be an topic for future work.

6.2   SLA Modeling and Description Languages

—- SLA attributes and its criteria are investigated for cloud computing by Alhamad et al. [10]. This study determined the individual SLA metrics for SaaS, PaaS and IaaS separately. Although specific metrics are presented based on their layer requirements, they never considered the SLAs relation and hierarchical nature of cloud computing.

Patel et al. [107] proposed the WSLA framework for SLA enforcement in cloud computing while this structure already proposed for SOA by Keller and Ludwing [96]. This framework is not enough adapted for cloud needs. Cloud computing nature and requirements are totally different

from SOA therefore SLA management systems cannot directly be brought from SOA.

Some other frameworks such as RESTful [88], SALmonADA [101], SLA@SOI [2] have put aside the hierarchical nature of cloud computing. They have directly applied the SLA structure in the cloud computing area from SOA and grid computing.

The SSV architecture proposed by Kertesz et al. [85]which contracted the SLA between an individual service provider and consumer after the negotiation process. However, they did not consider the hierarchical relation of SLAs in the cloud environment.

Although a research by Undheim et al. [138] attempted to deploy the on demand SLAs with QoS details on different levels, it followed the common SLA structure with stated shortcomings.

Different techniques of SLA negotiation including direct, broker and market based negotiations are investigated and compared by Grandits et al. [29] while they did not have enough attention to the related SLAs in negotiation process. So, the output of these techniques is the isolated agreed SLA.

——

NIST [94] has also pointed out the necessity of SLAs, SLA management, definition of contracts, orientation of monitoring on Service Level Objects (SLOs) and how to enforce them. However a clear definition of a reference of a specific format is missing. This is also the case with the Internet Engineering Task Force (IETF) [86]. Besides these approaches of the companies and organisations there are further efforts to develop and to realize cloud management architectures and systems. A basic discussion can be found in the book from Wieder et.al. [66] mainly concerned about SLA definitions and negotiations.

In 2003 IBM developed the Web Service Agreement Language (WSAL) [96] which is still available in the Version 1.0 that dates back to the same year. It has not been further developed since. WSAL focused on performance and availability metrics. It seriously lacks expression therefore it is not powerful enough. The required flexibility is also missing which is needed for dynamical changes at run time. It is closely connected to their XML schema and not very useful to determine conclusion. WSAL has been mainly developed for Web services, its usage in other fields is questionable. It shows significant shortcomings regarding content as it focusses mainly on technical properties. The structural requirements, however, are met as discussed in Spillner et.al. [128].

The WS-Agreement (WS-A) was developed by the Open Grid Forum in the year 2007 (Version 1.0) The last update which was based on the work of the European SLA@SOI project was done in 2011. Its advantages are the expandability and the adaptability which is, on the other hand, also one of its greatest disadvantages because it has not been specified in details by Kearney [82]. It is based on technical transformation, the structural transformations, however, have not been taken into account.

The Foundation of Self Governing ICT Infrastructures (FOSll)-Project [51] is another research project which aims at the usage of autonomic principals for information and communication

systems. Self determining infrastructures should be realized and made available through cloud based services. Within the LoM2HiS autonomic SLA management is realized by translation of system parameters to abstract KPIs and SLOs [23]. The SLA specification is based on WSAL. The Texo project [53] attempts within the THESEUS research program to realize a conception of service descriptions, contract management, end to end marketplaces and monitoring from a business perspective. In addition the development and use of WS-A based SLAs are needed. Looking at the cloud interfaces that describe the management of resources within the cloud, it becomes obvious that they are exclusively designed with the purpose of system monitoring. They do not provide a direct monitoring of SLAs. Therefore the placement of machine readable SLAs is extremely difficult. This is also the case with existing monitoring tools.

SLA handling in clouds, resulting from the EU project OPTIMIS, is discussing negotiating and creating Service Level Agreements between infrastructure providers and service providers [91]. Although it is enhanced within the SLA@SOI project [2] its development is unclear because the SLA@SOI project develop its own format SLA@SOI SLA(T) [3] and supported by the European IT industry. A comprehensive project result has been published on their web page. No independent analysis of the advantages or disadvantages of the SLA(T) format is available at the moment. It provides all structural requirements of a SLA and it has the greatest intersection with regard to content. SLA(T) is the basis of the proposed approach in this paper. Further it should be accentuated that a meta model SLA* [1] is defined which simplifies the extension and adaptability for SLA(T).

6.3 Performance Prediction

Neural Networks are widely used in forecasting problems. One of the earliest successful application of ANNs in forecasting is reported by Lapedes and Farber [90]. They used a feed-forward neural network with deterministic chaotic time series generated by the Glass-Mackey equation, to predict such dynamic non-linear systems. Artificial Neural Networks are proven universal approximators [71][70] and are able to forecast both linear [146] and non-linear time series [145]. Adya and Collopy investigated in the effectiveness of Neural Networks (NN) for forecasting and prediction [9]. They came to the conclusion that NN are well suited for the use of prediction, but need to be validated against a simple and well-accepted alternative method to show the direct value of this approach. Since forecasting problems are common to many different disciplines and diverse fields of research, it is very hard to be aware of all the work done in this area. Some examples are forecasting applications such as: temperature and weather [89][136][117], tourism [108], electricity load [105][68], financial and economics [21][99][81][65] and medical [141][14] to name a few. Zhang, Patuwo, and Hu [145] show multiple other fields where prediction by ANN was successfully implemented.

Similar research with different focus has been conducted in the past for the use of machine

learning in cloud environments. Prevost et al. used a Neural Network (NN), as well as a Linear Predictor [111] to anticipate future workloads by learning from historical URL requests. Although both models were able to give efficient predictions, the Linear Predictor was able to predict more accurately. Li and Wang proposed their modified Neural Network algorithm nn-dwrr in [93]. The application of this algorithm led to a lowered average response time compared to application of traditional capacity based algorithms for scheduling incoming requests to VMs. In similar research Hu et al. [72] have shown that their modification of a standard Support Vector Regression (SVR) algorithm can lead to an accurate forecasting of CPU Load what can be used to achieve a better resources utilization. Another algorithm, which is renowned for providing good results in similar scenarios, is Linear Regression. Although the results are often weaker compared to Neural Networks or Support Vector Machines (SVM) in cases of workload prediction [17] [75], the fast training and deployment time of models built with Linear Regression should not be underestimated. Those examples show that there are a variety of optimization challenges in cloud environments which can be tackled by applying machine learning algorithms. What separates the current work from previous research is a detailed examination of specific characteristics of three different machine learning algorithms and presenting the results in a visual way. The choice to evaluate Neural Networks, Support Vector Machines and Linear Regression was made because those algorithms earned promising results in previously conducted research.

6.4 Scheduling Mechanisms

Most commonly found scheduling approaches for the Cloud, like proposed by Mao and Humphrey [98] or Panda et al. [104], aim towards resource optimization based task scheduling. Therefore, cloud instances are regarded as computing tasks that have to be processed within certain resource constraints. However, this basic view is only suitable for cloud operations with already known tasks or performance requirements. In the real cloud environment, however, these are usually of no importance since cloud customers either do not know them or they are highly variable. The approach shown seems to have more applicability in grid computing.

Haizea [125] is a resource manager or resource scheduler. Haizea can manage a set of computers (typically a cluster), allowing users to request exclusive use of those resources described in a variety of terms, such as "I need 10 nodes, each with 1 GB of memory, right now" or "I need 4 nodes, each with 2 CPUs and 2GB of memory, from 2pm to 4pm tomorrow". Haizea provides reservation and deadline sensitive type of leases along with the traditional immediate and best effort policies [126] [124] [127]. Haizea uses simple allocation policies for deadline sensitive leases. It tries to find out a single slot of required time between startTime and endTime of the given lease which can allocate the requested resources. If it is unable to ?nd such a time slot, it rejects the lease.

6.5 SLA Enforcement

Boniface et al. [3] discuss dynamic service provisioning using GRIA SLAs. The authors describe provisioning of services based on agreed SLAs and the management of the SLAs to avoid violations. Their approach is limited to Grid environments. Moreover, they do not detail how the low-level metric are monitored and mapped to high-level SLAs.

Comuzzi et al. [33] define the process for SLA establishment adopted within the EU project SLA@SOI framework. The authors propose the architecture for monitoring SLAs considering two requirements introduced by SLA establishment: the availability of historical data for evaluating SLA offers and the assessment of the capability to monitor the terms in an SLA offer.

SLAs-LoM2HiS framework The LoM2HiS framework comprises two core components, namely host monitor and run-time monitor. The former is responsible for monitoring low-level resource metrics, whereas the latter is responsible for metric mapping and SLA violation monitoring. In order to explain our mapping approach we consider the Service Level Objectives (SLOs) as shown in Table 1, including incoming bandwidth, outgoing bandwidth, storage, and availability. As shown in Figure 1, we distinguish between host monitor and run-time monitor. Resources are monitored by the host monitor using arbitrary monitoring tools such as Ganglia [27]. Resource metrics include downtime, uptime, and available storage. Based on the predefined mapping rules stored in a database, monitored metrics are periodically mapped to the SLA parameters. An example of an SLA parameter is service availability Av, (as shown in Table 1), which is calculated using the resource metrics downtime and uptime and is defined by the following the mapping rule: As described in a previous work [14], we implemented a highly scalable framework for mapping Low Level Resource Metrics to High Level SLA Parameters (LoM2HiS framework) facilitating the exchange of large numbers of messages. We designed and implemented a communication model based on the Java Messaging Service (JMS) API [21], which is a Java Message Oriented Middleware API for sending messages between two or more clients. We use Apache ActiveMQ as a JMS provider that manages sessions and queues.

——

The LoM2HiS framework introduced by Emeakaroha et al. [**?**] as a part of FOSII [51] infrastructure. They tried to map the low level metrics into high level SLA by their monitoring method but the common SLA is used and they followed the centralized monitoring system [43]. The LoM2HiS monitoring framework is located in FoSII infrastructure which calculated the SLA attributes value based on the proposed metrics and formula [43]. In this framework, the SLAs consist of attributes, metrics and formulas have to be located in a central repository. Then, another monitoring system is used to measure the SLA validity.

— Another related work is DeSVi architecture by Emeakaroha et al. [44], it is designed based on LoM2HiS [42] framework which limited for only one data centre as an IaaS.

—- The QoSMONaaS approach [30] focused to the QoS monitoring as a service. Although this flexible approach is a third party SLA monitoring system, it still needs the external service for

SLA validating.

—— The violation reaction process is rarely discussed in the mentioned SLA monitoring frameworks. The most of monitoring systems just tried to report the detected violations [101] [33] or assess the penalty cost of service providers [140] [36].

However, an effective reaction process is needed to react against SLA violations. Some studies such as [62], [95] and [142] utilized the SLAs to manage the provider resources without any reaction against violations.

The FoSII and DeSVi frameworks introduced by Emeakaroha et al. [?][44]. They applied the monitoring, analysis, planning and execution loop (MAPE) model to enforce the self-healing cloud computing. This architecture has beneficial for cloud management but not for sudden violation reaction. In another study, the LoM2HiS [?] framework tried to predict the SLA violations by predefining the thresholds. Then, the alerts are notified into the internal knowledge component for possible adjustment in provider resource [35] however there were not any other reaction strategies.

On the other hand, LAYSI [22] has propagated the detected SLA violations into upper layer but its communication model was based on subscription between meta-negotiators, brokers and automatic service deployed. This reaction model was complex and followed the plain SLA document structure which is in contrast of selfhealing SLA.

This literature review has illustrated that the related works and proposed solutions did not suitably cover the cloud computing requirements. The most of related works have transmitted the SLA monitoring framework into cloud computing from SOA and grid computing. Although a few effective SLA monitoring and reacting mechanisms have presented, they have followed the common SLA structure yet. Having the selfhealing SLA in cloud computing, an effective selfmonitoring SLA and reaction mechanism are needed.

Slaws Specs EUROPEAN COMMISSION DIRECTORATE GENERAL COMMUNICATIONS NETWORKS, CONTENT AND TECHNOLOGY UNIT E2 - SOFTWARE AND SERVICES, CLOUD https://www.sla-ready.eu/sla-negotiation

6.6 Autonomic Computing

Autonomic computing refers to the ability of a distributed system to manage its resources with little or no human intervention. It's all about creating a system that can intelligently adapt to changing environments and needs without the user even knowing it. It was launched in 2001 by IBM to reduce the complexity of managing large distributed systems [69].

Autonomic computing in the area of cloud computing

Autonomic resource allocation [73] [27]

Autonomic self-testing [87]

Autonomic life-cycle management

CONCLUSION AND FUTURE WORK

7.1   Summary

This thesis addressed the problem of managing service execution in cloud infrastructures while meeting SLA constraints. SLA management functionalities are proposed which handle service instantiation, provisioning and termination in an autonomous fashion. In addition, the thesis employs further management actions and prediction algorithms in order to increase the profit by reducing infrastructure costs and preventing SLA violations.

SLAs and KPIs The current Cloud SLA landscape and developments towards SLA for cloud services have been presented. The general SLA management process and its adoption to the cloud computing model has bee presented. For this thesis a special SLA creation process has been implemented and it's SLO-A format has been presented, thus making it possible to write legally binding contracts in a machine readable way. Allowing customers of cloud services to specify exactly which service level they booked for their cloud service and where, how and when to check it.

SLA Automation This thesis has presented an environment for the autonomous management of service level agreements in cloud computing through the presented technology, architecture and prototypical implementation. Hereby the consumer is able to individually adjust the performance of his services and the resulting service levels using a predefined KPI catalog, where the resulting SLA is directly monitored for compliance by autonomous management, and possible violations are prevented adaptively by resource control algorithms. This new kind of resource management and service management increases the trust and efficiency of the cloud environment and enables to build more trust into cloud environment services.

SLA Compliance Mangement The SLA compliance management algorithms presented and

analysed here showed that the number of SLA violations could be reduced by already drastically by expanding the current methods of resource management in cloud environments. Furthermore, the SLA compliance became even better through the use of latest machine learning approaches which could predict the environmental variables of the system and the usage by the user. These innovative approaches make it possible to make better use of existing services, to better integrate new services into the system and to reduce the reliability and therefore the costs for the provider.

Resource Acquisition and Allocation This thesis includes initial proposals for resource acquisition and allocation. Based on scheduling procedures, the required resources are used as efficiently as possible and additional resources are provided in the event of a SLA violation. In order to increase the provider profit, this thesis has shown how packing algorithms can generate resource utilization close to potential optimum. In addition, it was considered that in a future version contracts with lower priorities or penalties would be exchanged for contracts with high priorities / penalties as soon as the required resources can no longer be provided to keep the penalties as low as possible.

7.2   Lessons Learned

While working on this thesis some interesting insights were gained via the experimental implementations. As shown by the fuzzy approach, even small improvements in resource allocation of cloud system can result in great improvements in service level compliance. Although the machine learning approaches can improve these results, it still remains questionable whether the extra effort justifies the result. Especially in the case of time-critical real-time systems, due to the fast reaction time of the system, prediction must be viewed critically. During my extensive work with cloud systems, it has been shown that simpler, but easy-to-use solutions are particularly more often chosen in corporate environments, but this does not reduce the interest in cutting edge solutions. Especially in the cooperation during the research project and the industry, it was repeatedly shown how important an accurate and legally correct description of SLAs and KPIs is. Through the resulting catalog of directives, we were able to return this knowledge back to practical application. The research on this work and the possibility of carrying out tests and measurements in a real environment and then working with the industry revealed several interesting insights. In fact, my work clearly showed that in contrast to developmental environments in the real world outages or SLA violations, things can happen much more frequently because of things that can not be steered directly through an automatism. Worse still, in today's complex multi-cloud environments, both internal and external factors play a big role on which one may not always be able to influence.

7.3 Future Work

Scalability The current implementation of the ASLAMaaS framework does not aim at scalability. Expanding the system with external cloud services such as Amazon EC2 or the Alibaba Cloud could enable the overall system to respond even more dynamically to the load of users. Both the acquisition of resources and the associated integration would have to be automated, as well as the dynamic management or precautionary planning of resources. For example, Amazon offers cheaper resources if they are booked in advance for a foreseeable period of time, which in turn can have a positive impact on costs. These extensions would extend the existing system by a further dimension in the planning and enable much greater dynamics. Simultaneously the move from one single resource provider and therefore only horizontal elasticity as means for improving performance, to an multi cloud environment with various elasticity in horizontal and vertical order as well as global distribution of resources. Exploiting this additional elasticity allows implementing further QoS assurance mechanisms. Regarding multi cloud environments, future work may consider checking prices and negotiating contracts with them to improve costs. In addition this would allow overcoming resource shortages and the utilization of cheaper resources.

On-The-Fly Profiling The profiling mechanism implemented profiles the service provider before provisioning. However, profiling data during provisioning is useful for calibrating according to unpredicted environment changes, e.g., network throughput degradation and unusual customer demands. An improvement to this issue is enabling to update the provider profiling while treating customer requests.

Renegotiation The SLA contracts considered in this work can be more flexible by including renegotiation (alteration) of established contracts. Supporting renegotiation would allow adapting the service provisioning to environment changes without violating SLAs.

Improved machine learning algorithms The field of machine learning has grown in the last time very rapidly. This may results in new technologies that improve the use of machine learning algorithms and make them more efficient. The expansion of the ML approaches could lead to future work being able to generate on the fly predictions about the state and the future use of the system not only more accurately but also much faster. As a result, the possibility of countering possible SLA violations already arises and thus achieving even higher SLA compliance. Especially the area tensor flow could be interesting here.

Prove of Concept for Holistic SLA Management Since the approach presented here for the holistic management of SLAs was presented purely computationally and through simulations, a prototypical implementation in a real cloud landscape would be desirable. Unfortunately, in order to get a realistic experience, this is a challenge for both the infrastructure and the test user.

[1]   SLA@SOI framework alpha.
       Online.

[2]   SLA@SOI projekt website.
       Online.

[3]   SLA@SOI: SLA model: The model.
       Online.

[4]   *MASCOTS 2000, Proceedings of the 8th International Symposium on Modeling, Analysis
       and Simulation of Computer and Telecommunication Systems, 29 August - 1 September
       2000, San Francisco, California, USA.* IEEE Computer Society, 2000.

[5]   BITKOM Cloud Monitor 2014, 2014.
       [Accessed: 18. 08. 2014].

[6]   Crisp Research AG Cloud Computing Trends 2015, 2015.
       [Accessed: 02. 10. 2015].

[7]   Microsoft Office 365, 2014.
       [Accessed: 12. 09. 2014].

[8]   Giuseppe Aceto, Alessio Botta, Walter De Donato, and Antonio Pescapè.
       Cloud monitoring: A survey.
       Computer Networks, 57(9):2093–2115, 2013.

[9]   M. Adya and F. Collopy.
       How effective are neural networks at forecasting and prediction? a review and evaluation.
       Journal of Forecasting - Special Issue: Neural Networks and Financial Economics, 17(5-
       6):481–495, September 1998.

[10]   M. Alhamad, T. Dillon, and E. Chang.
       Conceptual SLA framework for cloud computing.
       April 2010.

[11] Amazon.
AWS Simple Monthly Calculator, 2014.
http://calculator.s3.amazonaws.com/index.html?lng=de12. 09. 2014].

[12] Amazon EC2 Service Level Agreement, 2013.
[Accessed: 02. 05. 2015].

[13] Google Apps, 2014.
[Accessed: 12. 09. 2014].

[14] Carlos Arizmendi, Daniel A. Sierra, Alfredo Vellido, and Enrique Romero.
Automated classification of brain tumours from short echo time in vivo mrs data using
gaussian decomposition and bayesian neural networks.
Expert Systems with Applications, 41(11):5296–5307, 2014.

[15] HFU Cloud Research Lab Autonomic SLA Management as a Service Project Website, 2014.
[Accessed: 25. 07. 2014].

[16] ASLAMaaS Project.
Autonomic SLA Management as a Service Project - SLA Richtliniendokument für Cloud
Dienstleistungen, 2014.
http://www.wolke.hs-furtwangen.de/assets/files/ASLAMaaS-SLA-
Richtliniendokument.pdf [Accessed: 25. 07. 2014].

[17] Akindele A Bankole and Samuel A Ajila.
Predicting cloud resource provisioning using machine learning techniques.
2013.

[18] Salman A. Baset.
Cloud slas: Present and future.
SIGOPS Oper. Syst. Rev., 46(2):57–66, July 2012.

[19] A. Behl and K. Behl.
An analysis of cloud computing security issues.
Oct 2012.

[20] Thomas G. Berger.
Konzeption und Management von Service-Level-Agreements für IT-Dienstleistungen.
TU Darmstadt, 2005.

[21] Yevgeniy Bodyanskiy and Sergiy Popov.
Neural network approach to forecasting of quasiperiodic financial time series.
European Journal of Operational Research, 175(3):1357–1366, 2006.

[22] I. Brandic, V. C. Emeakaroha, M. Maurer, S. Dustdar, S. Acs, A. Kertesz, and G. Kecskemeti.
LAYSI: A Layered Approach for SLA-Violation Propagation in Self-Manageable Cloud Infrastructures.
July 2010.

[23] Ivona Brandic, Dejan Music, Philipp Leitner, and Schahram Dustdar.
VieSLAF Framework: Enabling Adaptive and Versatile SLA-Management.
GECON '09. Springer-Verlag, Berlin, Heidelberg, 2009.

[24] Jérémy Buisson.
Adaptation dynamique de programmes et composants parallèles.
PhD thesis, Thèse de doctorat, INSA de Rennes, 2006.

[25] Jérémy Buisson, Françoise André, and Jean-Louis Pazat.
Dynamic adaptation for grid computing.
Advances in Grid Computing-EGC 2005, pages 139–168, 2005.

[26] Nicholas G. Carr.
The Big Switch: Der grosse Wandel - Die Vernetzung der Welt von Edison bis Google.
Heidelberg: Mitp, ISBN 978-3-8266-5508-1, 1st edition, 2009.

[27] Emiliano Casalicchio, Daniel A. Menascé, and Arwa Aldhalaan.
Autonomic Resource Provisioning in Cloud Systems with Availability Goals.
CAC '13. ACM, New York, NY, USA, 2013.

[28] The Guardian-Naked celebrity hack: security experts focus on iCloud backup theory, 2014.
[Accessed: 23. 10. 2014].

[29] Ramnath K. Chellappa.
Intermediaries in cloud-computing: A new computing paradigm.
In INFORMS Dallas 1997, Cluster: Electronic Commerce, Dallas, Texas„ 1997.

[30] Giuseppe Cicotti, L Coppolino, Salvatore D, N.A. Antonio, and Luigi Romano.
How to monitor qos in cloud infrastructures: The qosmonaas approach.
International Journal of Computational Science and Engineering, 11:29, 01 2015.

[31] Cloud Security Alliance.
Security Guidance for Critical Areas of Focus in Cloud Computing v3.0, 2011.
https://downloads.cloudsecurityalliance.org/assets/research/security-guidance/csaguide.v3.0.pdf [Accessed: 25. 07. 2016].

[32] Google Compute, 2014.
[Accessed: 12. 09. 2014].

[33] M. Comuzzi, C. Kotsokalis, G. Spanoudakis, and R. Yahyapour.
Establishing and Monitoring SLAs in Complex Service Based Systems.
July 2009.

[34] Contabo.
Dedicated Server Hosting, 2014.
https://contabo.de [Accessed: 12. 09. 2014.

[35] Maxwell Cooter and Cloud Pro.
The Ultimate Guide to Cloud Computing.
Dennis Publishing, 2011.

[36] Amir Vahid Dastjerdi, Sayed Gholam Hassan Tabatabaei, and Rajkumar Buyya.
A dependency-aware ontology-based approach for deploying service level agreement moni-
toring services in cloud.
Softw. Pract. Exper., 42(4):501–518, April 2012.

[37] Dimension Data.
Comparing public cloud service level agreements, 2013.
[Accessed: 02. 05. 2015].

[38] Distributed Management Task Force.
Architecture for managing clouds.

[39] Google Docs, 2014.
[Accessed: 12. 09. 2014].

[40] Google Drive, 2014.
[Accessed: 12. 09. 2014].

[41] Amazon Elastic Compute Cloud (Amazon EC2).
2006.
[Accessed: 22. 06. 2014].

[42] V. C. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar.
*Low level Metrics to High level SLAs - LoM2HiS framework: Bridging the gap between
monitored metrics and SLA parameters in cloud environments.*
June 2010.

[43] Vincent Emeakaroha, Ivona Brandic, Michael Maurer, and Schahram Dustdar.
Cloud resource provisioning and sla enforcement via lom2his framework.
Concurrency and Computation: Practice and Experience, 25, 07 2013.

[44] Vincent Emeakaroha, Marco Netto, Rodrigo Calheiros, Ivona Brandic, Rajkumar Buyya, and Cesar DeRose.
Towards autonomic detection of sla violations in cloud infrastructures.
Future Generation Computer Systems, 28(7):1017 − 1029, 2012.
Special section: Quality of Service in Grid and Cloud Computing.

[45] Google App Engine, 2014.
[Accessed: 12. 09. 2014].

[46] Experton Group.
Experton-analyse: Wer f/"ult die cloud mit business services.
http://www.computerwoche.de/management/cloud- computing/1934027/ [Accessed: 17. 03. 2014].

[47] Kaniz Fatema, Vincent C Emeakaroha, Philip D Healy, John P Morrison, and Theo Lynn.
A survey of cloud monitoring tools: Taxonomy, capabilities and objectives.
Journal of Parallel and Distributed Computing, 74(10):2918–2933, 2014.

[48] Wang Fei and Li Fan-Zhang.
The design of an autonomic computing model and the algorithm for decision-making, volume 1.
2005.

[49] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, and E. Turrini.
QoS Aware Clouds.
July 2010.

[50] International Organization for Standardization ISO/IEC NP 19086-2 Information technology Cloud computing Service level agreement (SLA) framework and Technology Part 2: Metrics.
2014.
[Accessed: 22. 09. 2014].

[51] FOSII Team.
Foundations of self-governing ict infrastructures website.

[52] I. Foster, Yong Zhao, I. Raicu, and Shiyong Lu.
Cloud Computing and Grid Computing 360-Degree Compared.
IEEE, November 2008.

[53] Fraunhofer IAO.
Theseus Texo - Business Webs im Internet der Dienste, 2011.

https://www.e-business.iao.fraunhofer.de/de/projekte/beschreibung/theseus-texo.html [Accessed: 25. 07. 2015].

[54] S. Frey, C. Lüthje, V. Huwwa, and C. Reich.
Fuzzy Controled QoS for Scalable Cloud Computing Services.
CLOUD COMPUTING 2013, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization.
2013, pp. 150-155.

[55] S. Frey, C. Luthje, C. Reich, and N. Clarke.
Cloud QoS Scaling by Fuzzy Logic.
March 2014.

[56] S. Frey, C. Lüthje, R. Teckelmann, and C. Reich.
Adaptable Service Level Objective Agreement (A-SLO-A) for Cloud Services.
CLOSER 2013, pp. 457-462, SciTePress, 2013.

[57] Wei Fu and Qian Huang.
GridEye: A Service-oriented Grid Monitoring System with Improved Forecasting Algorithm.
GCCW '06. IEEE Computer Society, Washington, DC, USA, 2006.

[58] G. Cattaneo, M. Kolding, D. Bradshaw, G. Folco, IDC for the European Commission.
Quantitative estimates of the demand for cloud computing in europe and the likely barriers to take-up.
http://cordis.europa.eu/fp7/ict/ssai/docs/study45-d2-interim-report.pdf [Accessed: 12. 08. 2014].

[59] G. R. Gangadharan, G. Frankova, and V. D'Andrea.
Service license life cycle.
2007.

[60] Ganglia.
Ganglia Website.
https://ganglia.info/ [Accessed: 25. 01. 2018].

[61] Gartner Research Group - Hype Cycles 2014, 2014.
[Accessed: 25. 08. 2014].

[62] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira.
Workflow scheduling for SaaS / PaaS cloud providers considering two SLA levels.
April 2012.

[63] Sharon Eisner Gillett and Mitchell Kapor.

Coordinating the Internet.
The Self-Governing Internet: Coordination by Design, MIT Press, pp. 3 - 38, Cambridge, MA, USA, 1997.

[64] ISO/IEC SC 38 Study Group.
Jtc sc 38 study group report on cloud computing.
http://isotc.iso.org [Accessed: 17. 06. 2014].

[65] Erkam Guresen, Gulgun Kayakutlu, and Tugrul U. Daim.
Using artificial neural network models in stock market index prediction.
Expert Systems with Applications, 38(8):10389–10397, 2011.

[66] J. Happe, W. Theilmann, A. Edmonds, and K.T. Kearney.
Service Level Agreements for Cloud Computing.
Springer-Verlag, 2011.

[67] Peer Hasselmeyer, Bastian Koller, Ioannis Kotsiopoulos, Dean Kuo, and Michael Parkin.
Negotiating slas with dynamic pricing policies.
Proceedings of the SOC@ Inside'07, 2007.

[68] H.S. Hippert, C.E. Pedreira, and R.C. Souza.
Neural networks for short-term load forecasting: a review and evaluation.
Power Systems, IEEE Transactions on, 16(1):44–55, Feb 2001.

[69] Paul Horn.
Autonomic computing: Ibm\'s perspective on the state of information technology.
2001.

[70] Kurt Hornik.
Approximation capabilities of multilayer feedforward networks.
Neural Networks, 4(2):251–257, 1991.

[71] Kurt Hornik, Maxwell Stinchcombe, and Halbert White.
Multilayer feedforward networks are universal approximators.
Neural Networks, 2(5):359–366, 1989.

[72] Rongdong Hu, Jingfei Jiang, Guangming Liu, and Lixin Wang.
Kswsvr: A new load forecasting method for efficient resources provisioning in cloud.
2013.

[73] Ye Hu, Johnny Wong, Gabriel Iszlai, and Marin Litoiu.
Resource provisioning for cloud computing.
01 2009.

[74] IBM.
      The Virtualization Cookbook for IBM z/VM 6.3, RHEL 6.4, and SLES 11 SP3.
      IBM Redbooks, 2014.

[75] Md Toukir Imam, Sheikh Faisal Miskhat, Rashedur M Rahman, and M Ashraful Amin.
      Neural network and regression based processor load prediction for efficient scaling of Grid and Cloud resources.
      2011.

[76] Dropbox Inc., 2014.
      [Accessed: 12. 09. 2014].

[77] Gartner Inc.
      Cloud iaas slas can be meaningless http://blogs.gartner.com/lydia_leong/2012/12/05/cloud-iaas-slas-can-be-meaningless.
      http://blogs.gartner.com/lydia_leong/2012/12/05/cloud-iaas-slas-can-be-meaningless [Accessed: 12. 09. 2014].

[78] Intel Corp.
      What is holding back the cloud?
      Intel Corp., 2012.
      http://www.intel.com/content/www/us/en/cloud- computing/whats-holding-back-the-cloud-peer-research-report.html, [Accessed: 02. 08. 2014].

[79] ISO/IEC JTC 1/SC 38 Study Group.
      Study group report on cloud computing.
      Technical report, International Organization for Standardization, 2011.

[80] ITIL Foundation.
      Information Technology Infrastructure Library (ITIL).
      http://www.itil-officialsite.com/ [Accessed: 12. 09. 2014].

[81] Iebeling Kaastra and Milton Boyd.
      Designing a neural network for forecasting financial and economic time series.
      Neurocomputing, 10(3):215–236, 1996.
      Financial Applications, Part II.

[82] Keven T. Kearney, Francesco Torelli, and Constantinos Kotsokalis.
      Sla*: An abstract syntax for service level agreements.
      11th IEEE/ACM International Conference on Grid Computing, pages 217–224, 2011.

[83] Alexander Keller and Heiko Ludwig.
      The wsla framework: Specifying and monitoring service level agreements for web services.

Journal of Network and Systems Management, 11(1):57–81, March 2003.

[84] Jeffrey O Kephart and David M Chess.
      The vision of autonomic computing.
      Computer, 36(1):41–50, 2003.

[85] A. Kertesz, G. Kecskemeti, and I. Brandic.
      Autonomic SLA-Aware Service Virtualization for Distributed Systems.
      Feb 2011.

[86] B. Khasnabish, J. Chu, S. Ma, Y. Meng, N. So, and P. Unbehagen.
      Cloud reference framework.
      Technical report, Internet Engineering Task Force, 2010.

[87] T. M. King and A. S. Ganti.
      Migrating Autonomic Self-Testing to the Cloud.
      April 2010.

[88] Roland Kübert, Gregory Katsaros, and Tinghe Wang.
      A RESTful Implementation of the WS-agreement Specification.
      WS-REST '11. ACM, New York, NY, USA, 2011.

[89] G. Langella, A. Basile, A. Bonfante, and F. Terribile.
      High-resolution space–time rainfall analysis using integrated {ANN} inference systems.
      Journal of Hydrology, 387(3–4):328–342, 2010.

[90] A. Lapedes and R. Farber.
      Nonlinear Signal Processing Using Neural Networks: Prediction and System Modelling.
      Technical Report LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, NM, 1987.

[91] Andy Lawrence, Karim Djemame, Oliver Wäldrich, Wolfgang Ziegler, and Csilla Zsigri.
      Using service level agreements for optimising cloud infrastructure services.
      ServiceWave'10. Springer-Verlag, Berlin, Heidelberg, 2011.

[92] Fang Liu; Jin Tong; Jian Mao; Robert B. Bohn; John V. Messina; Mark L. Badger; Dawn M. Leaf;.
      NIST Cloud Computing Reference Architecture.
      National Institute of Standards and Technology, 2011.
      [Accessed: 24. 06. 2014].

[93] Chung-Cheng Li and Kuochen Wang.
      An SLA-aware load balancing scheme for cloud datacenters.
      2014.

[94] F. Liu, J. Tong, J. Mao, R. B. Bohn, J. V. Messina, M. L. Badger, and D. M. Leaf.
Nist cloud computing reference architecture.
Technical report, National Institute of Standards and Technology, 2011.

[95] H. Liu, D. Xu, and H. K. Miao.
Ant Colony Optimization Based Service Flow Scheduling with Various QoS Requirements in Cloud Computing.
Dec 2011.

[96] Heiko Ludwig, Alexander Keller, Asit Dan, Richard P. King, and Richard Franck.
Web Service Level Agreement (WSLA) Language Specification, v1.0, January 2003.

[97] Edson Manoel, Morten Jul Nielsen, Abdi Salahshour, Sai Sampath K.V.L., and Sanjeev Sudarshanan.
Problem Determination Using Self-Managing Autonomic Technology.
IBM Redbooks, 2005.

[98] M. Mao and M. Humphrey.
Auto-scaling to minimize cost and meet application deadlines in cloud workflows.
Nov 2011.

[99] Peter McAdam and Paul McNelis.
Forecasting inflation with thick models and neural networks.
Economic Modelling, 22(5):848–867, 2005.

[100] Daniel A Menascé.
Load testing, benchmarking, and application performance management for the web.
2002.

[101] C. Müller, M. Oriol, M. Rodríguez, X. Franch, J. Marco, M. Resinas, and A. Ruiz-Cortés.
SALMonADA: A platform for monitoring and explaining violations of WS-agreement-compliant documents.
June 2012.

[102] Richard Murch.
Autonomic computing.
IBM Press, 2004.

[103] Microsoft OneDrive, 2014.
[Accessed: 12. 09. 2014].

[104] Sanjaya K. Panda, Indrajeet Gupta, and Prasanta K. Jana.
Allocation-aware task scheduling for heterogeneous multi-cloud systems.

Procedia Computer Science, 50:176 – 184, 2015.
Big Data, Cloud and Computing Challenges.

[105] D.C. Park, M.A El-Sharkawi, II Marks, R.J., L.E. Atlas, and M.J. Damborg.
Electric load forecasting using an artificial neural network.
Power Systems, IEEE Transactions on, 6(2):442–449, May 1991.

[106] Animesh Patcha and Jung-Min Park.
An overview of anomaly detection techniques: Existing solutions and latest technological trends.
Computer Networks, Volume 51, Number 21, 2007.

[107] Pankesh Patel, Ajith Ranabahu, and Amit Sheth.
Service Level Agreement in Cloud Computing.
2009.

[108] Douglas C Pattie and John Snyder.
Using a neural network to forecast visitor behavior.
Annals of Tourism Research, 23(1):151–164, 1996.

[109] Mell Peter and Grance Timothy.
The NIST Definition of Cloud Computing,.
National Institute of Standards and Technology, 2011.
[Accessed: 24. 04. 2014].

[110] Microsoft Azure Cloud Platform, 2014.
[Accessed: 12. 09. 2014].

[111] John J Prevost, KranthiManoj Nagothu, Brian Kelley, and Mo Jamshidi.
Prediction of cloud data center networks loads using stochastic and neural models.
2011.

[112] Rackspace, 2014.
[Accessed: 12. 09. 2014].

[113] Rackspace Ltd - Compare Service Levels, 2015.
Website, http://www.rackspace.co.uk/managed-cloud/compare-service-levels, [Accessed: 25. 08. 2015].

[114] Rapidminer.
Rapidminer Website.
https://rapidminer.com/ [Accessed: 25. 01. 2016].

[115] Antonio Regalado.
Who Coined 'Cloud Computing'?
MIT Technology Review Business in the Cloud, 2011.
[Accessed: 24. 06. 2014].

[116] Juniper Research.
Cloud computing - consumer markets - strategies & forecasts 2014-2018, 2014.
[Accessed: 12. 08. 2014].

[117] Paul J. Roebber, Melissa R. Butt, Sarah J. Reinke, and Thomas J. Grafenauer.
Real-time forecasting of snowfall using a neural network.
Weather and Forecasting, 22(3):676–684, 2014/07/15 2007.

[118] Salesforce, 2014.
[Accessed: 12. 09. 2014].

[119] Manasha Saqib.
Cloud monitoring: A survey.
International Journal of Advanced Research in Computer Science, 8(2), 2017.

[120] Holger Schmidt.
Entwurf von Service Level Agreements auf der Basis von Dienstprozessen.
Herbert Utz München, 2001.
ISBN: 3-8316-0071-6.

[121] R. Scholderer.
Management von Service-Level-Agreements: methodische Grundlagen und Praxislösungen mit CobiT, ISO 20000 und ITIL.
Dpunkt.Verlag GmbH, 2011.

[122] Service Now.
KPI Library, 2013.
http://mirror42.com [Accessed: 12. 07. 2014].

[123] Amazon Web Services, 2014.
[Accessed: 12. 09. 2014].

[124] Sotomayor, B., Foster, I., Keahey, K., 2006.

[125] Sotomayor, B., Keahey, K.: Haizea.
Haizea website, 2011.
http://haizea.cs.uchicago.edu/ [Accessed: 25. 01. 2011].

[126] Sotomayor, B., Montero, R.S., Llorente, I.M., Foster, I., 2009.

[127] Sotomayor, B., Montero, R.S., Llorente, I.M., Foster, I., 2009.

[128] Josef Spillner, Matthias Winkler, Sandro Reichert, Jorge Cardoso, and Alexander Schill.
Distributed Contracting and Monitoring in the Internet of Services.
DAIS '09. Springer-Verlag, Berlin, Heidelberg, 2009.

[129] Jonathan Spring.
Monitoring cloud computing by layer, part 1.
IEEE Security & Privacy, 9(2):66–68, 2011.

[130] Jonathan Spring.
Monitoring cloud computing by layer, part 2.
IEEE Security & Privacy, 9(3):52–55, 2011.

[131] S. Srinivasan.
Cloud Computing Basics.
Springer New York, ISBN 978-1-4614-7698-6, 1st edition, 2014.

[132] Roy Sterritt and Mike Hinchey.
Autonomic computing-panacea or poppycock?
2005.

[133] R. Sturm, W. Morris, and M. Jander.
Foundations of Service Level Management.
Sams Professionals. Pearson Sams, 2000.
ISBN: 978-0-6723-1743-9.

[134] Wenhui Sun, Yue Xu, and Feng Liu.
The role of XML in service level agreements management, volume 2.
2005.

[135] S. Kumaraswamy T. Mather and S. Latif.
Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance.
O'Reilly Media, 2009.

[136] J.W. Taylor and R. Buizza.
Neural network load forecasting with weather ensemble predictions.
Power Systems, IEEE Transactions on, 17(3):626–632, Aug 2002.

[137] Ralf Teckelmann.
Service level objective agreements format zur adaption von dynamischen slas in cloudia.
Master's thesis, Hochschule Furtwangen University, 2012.

[138] A. Undheim, A. Chilwan, and P. Heegaard.
Differentiated Availability in Cloud Computing SLAs.
Sep. 2011.

[139] Vanson Bourne Technology Market Research.
Companies struggling with cloud performance.
http://servicemanagement.cbronline.com/news/cloud-performance-issues-costing-firms-600000-a-year-survey [Accessed: 12. 11. 2013].

[140] P. Varalakshmi, K. H. Priya, J. Pradeepa, and V. Perumal.
SLA with Dual Party Beneficiality in Distributed Cloud.
Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[141] Arso M. Vukicevic, Gordana R. Jovicic, Miroslav M. Stojadinovic, Rade I. Prelevic, and Nenad D. Filipovic.
Evolutionary assembled neural networks for making medical decisions with minimal regret: Application for predicting advanced bladder cancer outcome.
Expert Systems with Applications, 2014.

[142] Xiaoying Wang, Zhihui Du, and Yinong Chen.
An adaptive model-free resource and power management approach for multi-tier cloud environments.
Journal of Systems and Software, 85(5):1135 – 1146, 2012.

[143] Jonathan Stuart Ward and Adam Barker.
Observing the clouds: a survey and taxonomy of cloud monitoring.
Journal of Cloud Computing, 3(1):24, 2014.

[144] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif.
Sandpiper: Black-box and gray-box resource management for virtual machines.
Comput. Netw., 53(17):2923–2938, December 2009.

[145] Guoqiang Zhang, B. Eddy Patuwo, and Michael Y. Hu.
Forecasting with artificial neural networks: The state of the art.
International Journal of Forecasting, 14(1):35–62, 1998.

[146] Guoqiang Peter Zhang.
An investigation of neural networks for linear time-series forecasting.
Computers and Operations Research, 28(12):1183–1202, 2001.

[147] Qi Zhang, Lu Cheng, and Raouf Boutaba.
Cloud computing: state-of-the-art and research challenges.
Journal of Internet Services and Applications, 1(1):7–18, 2010.

[148] Zhenxing Zhao, Congying Gao, and Fu Duan.
       A survey on autonomic computing research, volume 2.
       2009.

[149] Dimitrios Zissis and Dimitrios Lekkas.
       Addressing cloud computing security issues.
       Future Generation Computer Systems, 28(3):583–592, 2012.

APPENDIX A - PUBLISHED PAPERS & OUTPUTS

Conference Papers

1. Stefan Frey, Claudia Lüthje, Christoph Reich, "**Key Performance Indicators for Cloud Computing SLAs**" EMERGING 2013 - The Fifth International Conference on Emerging Network Intelligence, October 2013, ISBN: 978-1-61208-292-9

2. Stefan Frey, Claudia Lüthje, Vitali Huwwa, Christoph Reich, "**Fuzzy Controlled QoS for Scalable Cloud Computing Services**" CLOUD COMPUTING 2013 : The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization, June 2013, ISBN: 978-1-61208-271-4

3. Stefan Frey, Claudia Lüthje, Ralf Teckelmann, Christoph Reich "**Adaptable Service Level Objective Agreement (A-SLO-A) for Cloud Services**" International Conference on Cloud Computing and Services Science - CLOSER May 2013, page 457-462, SciTePress

4. Stefan Frey, Claudia Lüthje, Christoph Reich, Nathan Clarke, "**Cloud QoS Scaling by Fuzzy**" IC2E 2014 Proceedings of the 2014 IEEE International Conference on Cloud Engineering, Pages 343-348, ISBN: 978-1-4799-3766-0, DOI: 10.1109/IC2E.2014.30

5. Stefan Frey, Simon Disch, Christoph Reich, Martin Knahl, Nathan Clarke"**Cloud Storage Prediction with Neural Networks** CLOUD COMPUING 2015, The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization,Pages 52-56, ISBN: 978-1-61208-388-9

6. Matthias Lermer, Stefan Frey, Christoph Reich, "**Machine Learning in Cloud Environments Considering External Information** IMMM 2016: The Sixth International Conference on Advances in Information Mining and Management (includes DATASETS 2016), Pages 11-17 ISBN: 978-1-61208-477-0

Research Reports

1. Stefan Frey, Claudia Lüthje, Christoph Reich, Michael Maier, Tobias Zutavern, Markus Streif, "**SLA-Richtliniendokument für Cloud Dienstleistungen**"
Project: Autonomic Service Level Agreements as a Service, May 2013,
http://www.wolke.hs-furtwangen.de/assets/files/ASLAMaaS-SLA-Richtliniendokument.pdf

2. Stefan Frey "**Der Einsatz von genetischen Algorithmen für das Scheduling von Cloud-Instanzen**" HFU Informatik Journal 2011/12, November 2011; ISBN: 978-3-00-035367-3; p:11-17;