Fuzzy Controled QoS for Scalable Cloud Computing Services

Stefan Frey, Claudia Lüthje, Vitali Huwwa, Christoph Reich
Furtwangen University
Cloud Research Lab
Furtwangen, Germany
{stefan.frey,claudia.luethje,vitali.huwwa,christoph.reich}@hs-furtwangen.de

***Abstract*—An important characteristic of cloud infrastructures is scalability on demand. A scalability service monitors performance load metrics and decides to scale up or down, by provision or revoke of cloud resources. This could guarantee Quality of Service (QoS) and enforce Service Level Objectives (SLOs). The approach of this paper shows that with additional imprecise information (e.g. expected daytime performance) the up and down scale mechanism of such an infrastructure can be improved and SLA violation can be avoided.**

***Keywords*—Cloud Computing; Scaling Service; Fuzzy Logic; SLA; QoS**

I. INTRODUCTION

Cloud computing offers customers resources on demand on a self-service basis and gives them access to a large pool of computational power and storage. Customers do not have to manage and maintain their own IT assets and get charged by cloud providers based upon the amount of resources used or reserved. The fly in the ointment is the minimal guarantees of Quality of Service (QoS) for the user's applications. It is common that big cloud providers like Amazon offer only rudimentary service guarantees, like for example a guarantee for 99,95% availability of their EC2 cloud service. In most cases providers do not give any performance guarantees at all. Cloud computing services, like the auto scaling service of Amazon [1], scale the capacity of virtual machines (VM) up or down automatically according to e.g. CPU utilization. Such a service controls the number of VMs to maintain the performance of a service that experiences hourly, daily, or weekly variability in usage. The architecture of such a setup can be seen in Figure 1 inside the blue dashed box. This obviously has the potential to guarantee Key Performance Indicators (KPIs) indirectly but KPIs such as e.g. request response time which are typical Service Level Objectives (SLOs) in a Service Level Agreement (SLA) are not controlled directly.

Therefore, SLA violations can happen especially due to peak demands, caused by all kind of reasons (e.g. product launches, political statements, service advertisement, weather changes, etc.), and the up scaling delay of the infrastructure (e.g. VM start time, LB reconfiguration, infrastructure limits, and economical limits to prevent extraordinary costs, etc.). Other reasons for not matching the SLA guarantees could be limitations, like the maximum number of VMs or non-ideal load balancing algorithms, which are not considered in the approach of this paper. Decent scaling is very important, because the if scale down happens to early SLA violations occure and if its set to late the customer will pay for resources that are not utilized.

To minimize the number of SLA violations and to guarantee the QoS, an behaviour, load and performance prediction model is needed. If one could predict the usage of an service, looking ahead further than the infrastructure delay time, one could guarantee the QoS for that specific service.

The rest of the paper is organized as follows. In Section II the related research efforts are discussed. Then a detailed description of the problem of QoS in cloud computing can be found in Section III. In Section IV, the specific approach using fuzzy logic for controlling the scalable cloud service is introduced. The proof of concept is reported in Section V. Finally a conclusion is drawn and future work is suggested in Section VI.

II. RELATED WORK

Since 2009 many teams are working on the problem to improve the QoS for cloud computing. Armstrong and Djemame tried to transfer the technologies of QoS from grid computing to cloud computing as discussed in the paper "Towards quality of service in the cloud" [2]. The paper of Rochwerger et al. [3] discuss the funded project RESERVOIR, in which pooled resources handle peaks and slopes of resources.

Another interesting appraoch is the Q-Clouds framework described by Nathuji et al. [4]. This framework for the management of cloud servers enables the possibility to apply and control QoS. The introduced Q-states provide the possibility for users to define certain metric limits of SLOs, based on a cost model. The more the customer is ready to pay, the less likely is a SLA violation. The controller component uses a MIMO (multi-input, multi output) model for the calculation VM resources. So an input vector is defined by the platform controller. Based on that the output vector delivers the predicted QoS values. Unlike to the approach of this paper they basically use infrastructure metrics (e.g. performance, memory, etc.) to control the QoS.

The important next steps in the QoS for cloud computing were developed by Ferretti/Ghini/Panerieri [5]. Their paper presents an architecture, which provides cloud resources dynamically. The developed middleware tries to avoid SLA violations with the same use case as presented in this paper. Therefore they split the problem into three components: The

 ISBN: 978-1-61208-271-4

host platform is an dynamic configuration, that should guarantee the requirement of the SLA. The monitoring component is checking the host platform and its application to display changes in the configuration and violation of the QoS requirements. The third component is responsible for the dispatching and the load balancing. This component tries to keep the required QoS. It is implemented in an intelligent way, to distribute the available resources. It is distinct to load balancing for requests and sessions. Different to this paper is that the load balancer takes over the monitoring of the SLAs instead of a separate module. Further the only metric that is used is the actual request response time. Compared to our approach, there can be considered mean value, derivation value, imprecise information and admin control information.

Many recent works deal with computing resource allocation in clouds focusing specific management objectives, such as energy efficiency [6], fairness [7], economic fitness [8], and service differentiation [9]. All of the above works however deliver placement solutions. They do not consider the problem of controlling a load balanced, scalable Cloud service.

Related research can be found in the area of forecasting the load of electrical power in [10] and [11] where they use social, economic, and weather condition factors. To achieve QoS guarantees this paper uses such additional factors as well.

III. QoS IN CLOUD COMPUTING

A Service Level Agreement (SLA) is a contract between customer and provider, that specifies service performance properties. These properties are called Service Level Objectives (SLOs), which contain metrics called Key Performance Indicators (KPIs) and the specific value to be guaranteed. These performance metrics should be guaranteed over a relatively long time interval and if a metric is violated commonly penalty costs may have to be paid to the customer by the provider.

Compared to traditional data centers it is easier to guarantee QoS in cloud computing data centers, because of the possibility to automate infrastructure administration and added value services such as auto scaling. Today's virtualization technologies allow dynamic provisioning of virtual machines (VM), networks, storage, etc. Therefore a completely automatic, adaptable customer infrastructure is on the horizon to react in real time to load changes.

Especially a scalable infrastructure can easily be provisioned in the cloud service model Infrastructure as a Service (IaaS), that allows automatic up/down scaling according the actual load. This is a big step towards the possibility to guarantee KPIs like the service request response time, that is a widely used Key Performance Indicator and a common Service Level Objective (SLO).

The approach presented in this paper improves the up and down scaling by using additional information, like the expected load at a certain daytime in the future, expected increase at a specific future day because of special events, etc.. With the additional information we trie to forecast the load and therefore allow a better pre-acting up or down scale of the infrastructure if needed.

IV. FUZZY CONTROL TO IMPROVE QoS CONTROL

Most approaches consider infrastructure sensor data like bandwidth, request/response time, CPU usage, memory usage, etc. to control the scaling infrastructure as seen in Fig. 1 dashed box. The approach of this paper is to use additional, often imprecise information (e.g. weather) to improve the management to meet 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 (service administrator) in the form of non-numeric linguistic variables (e.g. age: young/old). These fuzzy inputs are used in the fuzzy control system, that uses expert knowledge to inference a fuzzy output. After defuzzifying this output to a crisp value, then this controls the overall scale system 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 scaled 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.

In the next Subsection IV-A the architecture of the fuzzy scaling cloud service is described, followed by subsection IV-B, discussing monitoring parameters, which will show the wide variety of information to improve the cloud scaling service. The last subsection IV-C presents the fuzzy control module.

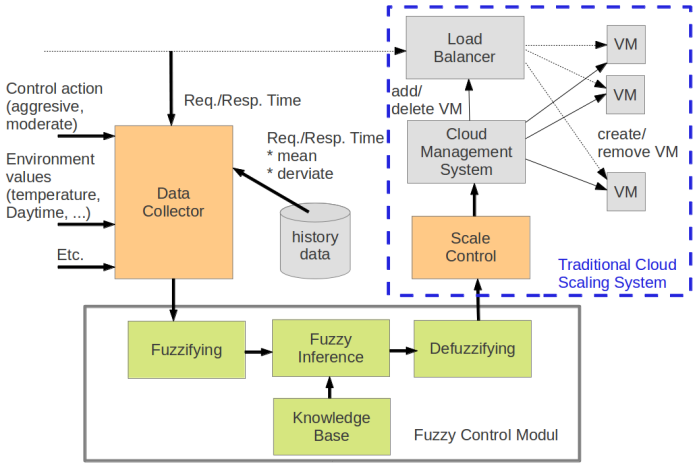

Fig. 1. Fuzzy Controlled Scaling Architecture

A. Fuzzy Controled Scaling Architecture

Figure 1 shows the architecture for a load balanced service by automatically scaling up/down the infrastructure by starting/stopping 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.

All collected data is input data to the *Fuzzy Control Module* where the data is fuzzified, results 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. This module generates XML-RPC calls to the *Cloud Management System*.

B. Information Factors for Control

The relevant information to improve the up/down scaling can be categorized into monitoring data: infrastructure, historic infrastructure, time-dependent, and service-dependent sensor data described in the following paragraphs in more detail.

a) Infrastructure Sensor Data: Table I lists factors that can mostly be monitored using sensors placed in various locations in the cloud infrastructure. KPIs, like request response time can easily measured at the load balancer (LB). Cloud specific parameters, like start time of VMs, can be aquisitioned at the cloud management system. If user service request types should be categorized (typically a imprecise parameter), it is best to ask the user admin of the cloud resource.

TABLE I. INFRASTRUCTURE SENSOR PARAMETER

Parameter	Example	Cloud Source
KPI	req./resp	load balancer
cloud specific indicators	VM start time, bandwidth	cloud management system
request type	long running req.	user
...	...	...

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.

b) History Infrastructure Sensor Data: Table II lists parameter that have been previously collected in a history data base. The purpose is to calculate values like, mean values, derivation values, etc. These statistical data can be good indicators to improve the LB management.

TABLE II. HISTORY INFRASTRUCTURE SENSOR PARAMETER

Parameter	Example	Source
derivation KPI	derivation req./resp	history DB
mean value KPI	req./resp. mean value	history DB
...	...	...

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 try to snowy condition makes it more likely, that the load of such a service is higher.

TABLE III. TIME-DEPENDENT SENSOR PARAMETER

Parameter	Example	Source
daytime	end of work	user input
weekday	Saturday	calendar
holiday	Christmas	country holiday cal.
product events	new iPhone	user input
...	...	...

c) Time-Dependent Sensor Data: Table III lists parameter that can influence the infrastructure management at a predefined time.

The knowledge of the typical weekly usage for an service (see Fig. 2) can be modelled and therefore the decision to scale up or down strongly or weekly depending whether the change is high or not.

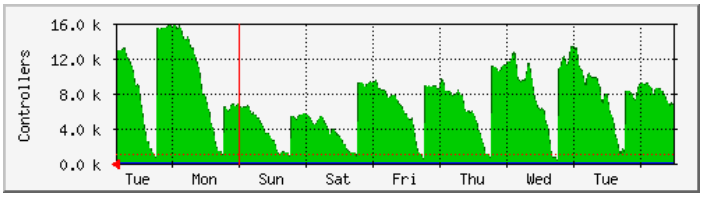

Fig. 2. Example: Weekly Load of the HFU Learning Management Platform

d) Service-Dependent Sensor Data: Table IV lists parameter 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, modelled 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.

TABLE IV. SERVICE-DEPENDENT SENSOR PARAMETER

Parameter	Example	Source
politic	EU summit	news ticker
market price	Facebook share	exchange feed
gossip	new Facebook mobile	news ticker
service importance	control behaviour (moderate/aggressive)	user
...	...	...

C. Fuzzy Control Module

The *Fuzzy Control Module* consists of four main fuzzy control processes represented by the four sub-modules respectively (see Fig. 1). 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 converted to the number of VM to started or stopped.

It follows a closer look at the 3 processes fuzzification, fuzzy inference and defuzzification.

 ISBN: 978-1-61208-271-4

Fuzzification: Fuzzification is the process of decomposing the input data into fuzzy sets, with trapezoidal shaped membership functions. Figure 4 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 modelled 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
   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 used in this approach and is defined as follows:

$$x^* = \frac{\int \mu_i(x) x dx}{\int \mu_i(x) dx} \quad (1)$$

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.

V. Proof of Concept by Simulation

In this section we discuss and evaluate the simulation results. The objective of the assessment was to verify whether or not our approach will ensure QoS for a cloud service better than conventional procedures. Hereafter we give a short introduction in our *Simulation Environment* (see section V-A) followed by the main features of our *Simulation Scenarios* (see section V-B), thereafter we discuss the results that we have obtained during several tests.

A. Simulation Environment

For feasibility testing, we created an simulation environment to be capable of validating the general fuzzy controlled scaling architecture proposed in this paper. The simulator therefore consists of four major components. Firstly, a request generator module, which simulates the generation of requests from an application to the cloud service. Here should be stated, that in our simulation requests are generated with an static workload. The second module is the load balancer which receives the generated requests of the request generator and distributes them to the pooled virtual machines. Here, the time is measured from the generation of the request till the execution inside a virtual machine is completed. This request response time then is checked by the scaler module, which decides either based on the fuzzy or the conventional rules whether to scale the service up, down or wait. The conventional rule set is an simple boundary system, where when the measured average request response hits the upper boundary a virtual machine gets started or when the lower boundary is hit a virtual machine is stopped. In between both boundaries the scaler waits.

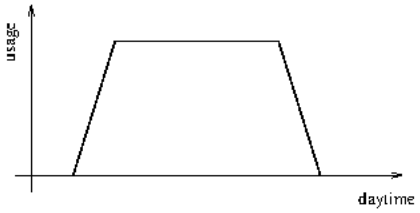

Fig. 3. Expected Load During Daytime

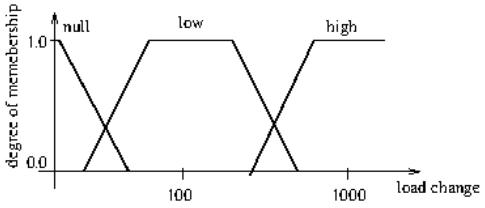

Fig. 4. Input Fuzzy Set for Load Deviation

The fuzzy set uses the same boundaries, but as an additional decision factor, a prediction based on expert knowledge is used.

Figure 3 shows the simplified load of an service during daytime. Based on such knowledge an expert specifies whether the load will be increasing at an *high*, *regular* or *low* rate. In case of an *high* prediction the fuzzy scaler generally scales up faster, which means it starts virtual machines on a lower load and additionally starts up to two virtual machines based on the load. Additionally it will scale down later, keeping a higher pool of available virtual machines. The *regular* prediction equals the conventional rule set, therefore resulting in essentially the same behavior. A *low* prediction, is in principle a reversed *high* prediction, which will change the behavior into generally scale up later and scale down faster. And similar to the *high* it is allowed to stop up to two virtual machines at once.

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 upscaling and at 1000ms by downscaling one virtual machine 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.

 ISBN: 978-1-61208-271-4

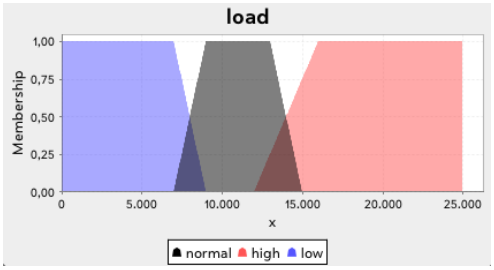

Fig. 5. Responsetime Fuzzy Sets

Figure 5 above shows the corresponding fuzzy set, where a load of below 9,000ms is considered *low* and above 15,000ms is *high*. In between stretches the range of *normal* load. These fuzzy values are combined with the fuzzy prediction values to create the set of fuzzy rules. To determine the suitability of the procedure presented, different scenarios have been created and tested with and without the fuzzy control mechanism. Following the scenario with the specific pre-conditions and characteristics is described and the obtained results a presented.

B. Simulation Scenarios

In the scenario the number of generated requests are increased rapidly and kept on a high level for an minute then to rapidly fall again. Figure 6 below shows the generated load graph for this scenario and the settings are shown in Table V.

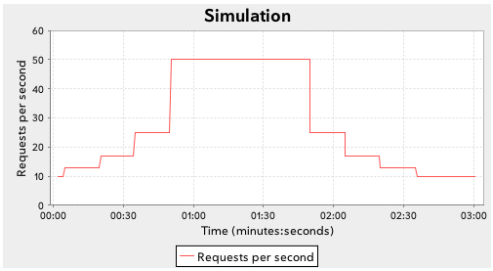

Fig. 6. Generated Load Scenario 1

This scenario simulates a peak load which happens when a service is facing an sudden demand. Such as accessing the canteen online menu just before the lunch break. Peak loads represent a problem in the real world, as countermeasures are most difficult.

TABLE V. SCENARIO 1 VALUES

Parameter	Value
min VMs	2
max VMs	10
runtime	180s

Figure 7 shows the simulation results with the conventional rules. Here it can be seen that the simulation begins with the minimum of 2 virtual machines in the pool. Until about 35s in the simulation the load is low enough for this two machines to cope with. After this point the average response time is rising slowly up until 50s where the load gets increased more. From this point, the response time increases sharply, until the first boundary limit of 15,000ms is hit and an additional virtual machine gets started.

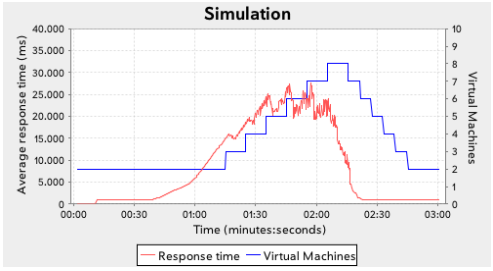

Fig. 7. Scenario 1 Conventional Results

The start of another VM is just not enough to improve the response time significant. Throughout the simulation up to 8 VMs are running simultaneously to manage the load. Comparing these results with the fuzzy controled results, where the prediction is set to *low*, shown in figure 8, it becomes clear that they are pretty similar. This is because with an *low* prediction, the limits for the up scale are corresponding to those of the conventional rule. Therefore the regulation starts on the same load adding VMs. When switching off VMs, the fuzzy scaler depending on the load cuts off two VMs. This behavior has however no effect on the in this simulation already sinking response time, but it could save resources and money in real life situations. In both cases the QoS limit of 20,000ms is exceeded.

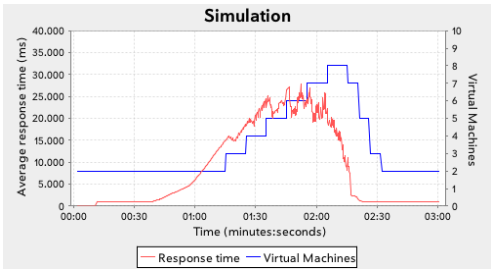

Fig. 8. Scenario 1 Fuzzy Low Prediction Results

The large fluctuations seen at the peak of the load can be explained by the forming the average response time. The individual values between newly started and already longer running VMs vary widely because of the waiting time of the processing packets in the different VMs input queues.

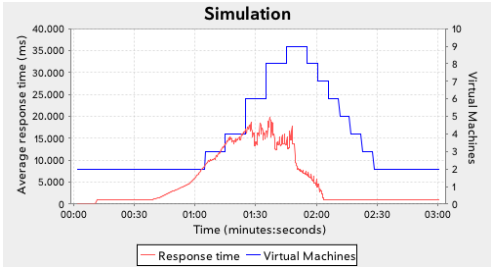

Fig. 9. Scenario 1 Fuzzy High Prediction Results

Figure 9 shows the results for the simulation with the fuzzy scaler and an *high* prediction. Compared to the other two tests

we can clearly see that the QoS limit of 20,000ms is respected this time. In general, we see that the response time runs in a similar but shallower curve. This can be attributed to the stronger up scale of starting two VMs simultaneously. This marginal difference is sufficient to prevent the response time from increasing over the QoS limit. The earlier intervention, which is already engaged at a normal load, prevents the requests from accumulating in the input queues of the VMs. Overall, though, more resources are used than in the other tests.

A comparison between the conventional rules and fuzzy scaler with *regular* prediction is not necessary, because these two sets are the same and thus generate the same results.

Over the running of all tests it has show that in all the scenarios considered, the fuzzy scaler is beneficial. Although this scaler uses in the *high* prediction more resources, for which some could argument it will cost more money, is the benefit in comparison greater, since a service in where less resource are needed but has no decent response times makes no sense to use. The *low* prediction did not improve the archived response time but releases the allocated resources faster than the conventional rules, therefore making it more economical. By the above presented tests it could be shown that by simple means, such as a fuzzy rule set and knowledge in form of an prediction, the response time could be improved or resources could be saved.

VI. Conclusion and Future Work

The goal of this paper was to show how a common cloud computing scaling service could be enabled to guarantee QoS parameters. Especially the KPI, request-response-time, has been the focus. The extended QoS provisioning architecture with an fuzzy control module has been delineated. A detailed description of possible new information to improve the scaling control system has been discussed. The proof of concept chapter showed that violation of SLAs could been avoided.

Future work is to proof this results within a real test environments and to develop an easy to use user interface. This shall allow users to specify imprecise information input and expert knowledge. Additionally the expansion to other QoS parameters, and further fuzzy input data has to be examined.

Acknowledgment

This research is supported by the German Federal Ministry of Education and Research (BMBF) through the research grant number 03FH046PX2.

References

[1] Amazon auto scaling service. Online. Retrived 01.2013. [Online]. Available: http://aws.amazon.com/autoscaling/

[2] D. Armstrong and K. Djemame, “Towards quality of service in the cloud,” 2009, pp. 226 – 240.

[3] B. Rochwerger, A. Galis, E. Levy, J. Caceres, D. Breitgand, Y. Wolfsthal, I. Llorente, M. Wusthoff, R. Montero, and E. Elmroth, “Reservoir: Management technologies and requirements for next generation service oriented infrastructures,” in *Integrated Network Management, 2009. IM ’09. IFIP/IEEE International Symposium on*, 2009, pp. 307 –310.

[4] R. Nathuji, A. Kansal, and A. Ghaffarkhah, “Q-clouds: managing performance interference effects for qos-aware clouds,” in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys ’10, New York, NY, USA, 2010, pp. 237–250.

[5] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, and E. Turrini, “QoS-Aware Clouds,” in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, ser. CLOUD ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 321–328.

[6] C.-T. Yang, K.-C. Wang, H.-Y. Cheng, C.-T. Kuo, and W. C. C. Chu, “Green power management with dynamic resource allocation for cloud virtual machines,” in *Proceedings of the 2011 IEEE International Conference on High Performance Computing and Communications*, ser. HPCC ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 726–733.

[7] F. Wuhib, R. Stadler, and M. Spreitzer, “A gossip protocol for dynamic resource management in large cloud environments,” vol. 9, no. 2, 2012, pp. 213–225.

[8] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, “Autonomic resource management in virtualized data centers using fuzzy logic-based approaches,” vol. 11, no. 3. Hingham, MA, USA: Kluwer Academic Publishers, Sep. 2008, pp. 213–227.

[9] J. Rao, Y. Wei, J. Gong, and C.-Z. Xu, “DynaQoS: model-free self-tuning fuzzy control of virtualized resources for qos provisioning,” in *Proceedings of the Nineteenth International Workshop on Quality of Service*, ser. IWQoS ’11. Piscataway, NJ, USA: IEEE Press, 2011, pp. 31:1–31:9.

[10] Y.-M. Wi, S.-K. Joo, and K.-B. Song, “Holiday load forecasting using fuzzy polynomial regression with weather feature selection and adjustment,” vol. 27, no. 2, may 2012, pp. 596 –603.

[11] K.-B. Song, S.-K. Ha, J.-W. Park, D.-J. Kweon, and K.-H. Kim, “Hybrid load forecasting method with analysis of temperature sensitivities,” vol. 21, no. 2, may 2006, pp. 869 – 876.

 ISBN: 978-1-61208-271-4