Systematically Deriving Quality Metrics for Cloud Computing Systems

Matthias Becker* Sebastian Lehrig† Steffen Becker†
{matthias.becker|sebastian.lehrig|steffen.becker}@{*uni-paderborn|†informatik.tu-chemnitz}.de
*Heinz Nixdorf Institute
University of Paderborn, Paderborn, Germany
†Software Engineering Chair
Chemnitz University of Technology, Chemnitz, Germany

ABSTRACT

In cloud computing, software architects develop systems for virtually unlimited resources that cloud providers account on a pay-per-use basis. Elasticity management systems provision these resources autonomously to deal with changing workload. Such changing workloads call for new objective metrics allowing architects to quantify quality properties like scalability, elasticity, and efficiency, e.g., for requirements/SLO engineering and software design analysis. In literature, initial metrics for these properties have been proposed. However, current metrics lack a systematic derivation and assume knowledge of implementation details like resource handling. Therefore, these metrics are inapplicable where such knowledge is unavailable.

To cope with these lacks, this short paper derives metrics for scalability, elasticity, and efficiency properties of cloud computing systems using the goal question metric (GQM) method. Our derivation uses a running example that outlines characteristics of cloud computing systems. Eventually, this example allows us to set up a systematic GQM plan and to derive an initial set of six new metrics. We particularly show that our GQM plan allows to classify existing metrics.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Performance measures*; D.2.11 [**Software Engineering**]: Software Architectures—*Languages*

Keywords

cloud computing; scalability; elasticity; efficiency; metric; SLO; analysis; GQM

1. INTRODUCTION

In cloud computing, software architects develop applications on top of compute environments being offered by cloud providers. For these applications, the amount of offered resources is virtually unlimited while elasticity management systems provision resources autonomously to deal with changing workloads. Furthermore, providers bill provisioned resources on a per-use basis [1]. As a consequence of these characteristics, architects want their applications to use as few resources as possible in order to save money while still maintaining the quality requirements of the system. Quality properties that focus directly on these aspects are scalability, elasticity, and efficiency [9].

These quality properties need to be quantified for requirements engineering and software design analysis by means of suitable metrics. For instance, cloud consumers and cloud providers need to negotiate service level objectives (SLOs), i.e., metrics and associated thresholds [7]. Such SLOs have to consider characteristics like changing workloads ("how fast can an application adapt to a higher workload?") and pay-per-use pricing ("how expensive is serving an additional consumer?"). However, no established metrics for requirements/SLO engineering and software design analysis exist. Current metrics assume knowledge of implementation details as they focus on the application at run-time [9] and lack a systematic derivation making such limitations explicit.

In literature, classical performance-oriented metrics [4] like response time and throughput are insufficient for situations relevant for cloud computing applications. First, they do not take changing workloads into account, e.g., metrics to describe reaction times to system adaptations are missing. Second, the degree to which systems match resource demands to changing workloads cannot be quantified. More recent work [9] proposes metrics for such characteristics that assume knowledge of implementation details like resource handling. These metrics are inapplicable when such knowledge is unavailable, e.g., in early design phases and for SLOs.

To cope with these lacks, we systematically derive an initial set of scalability, elasticity, and efficiency metrics using the goal question metric (GQM) method [15]. First, we illustrate the characteristics of cloud-aware applications using a running example scenario and, subsequently, derive metric candidates from this scenario. Second, by generalizing our metric candidates, we develop a first set of six metrics. Third, we show that our GQM plan allows to systematically classify existing metrics and makes their limitations explicit.

This paper contributes our systematic GQM plan, including classifications of our six and related work metrics.

This short research paper is organized as follows. Section 2 gives definitions of the considered quality properties and Sec. 3 introduced the running example system and its requirements. The system is implemented as cloud-aware application. In Sec. 4, we systematically derive a set of new metrics using the GQM method. In Sec. 5, we put our metrics in relation to existing metric proposals in literature and classify these metrics using our GQM plan in Sec. 6. Finally, Sec. 7 concludes the paper and highlights future work.

ICPE'15, Jan. 31–Feb. 4, 2015, Austin, Texas, USA.

http://dx.doi.org/10.1145/2668930.2688043

2. DEFINITIONS

Because we distinguish scalability, elasticity, and efficiency throughout our paper, we first give the definitions of these properties based on the work of Herbst et al. [9]: "**Scalability** is the ability of the system to sustain increasing workloads by making use of additional resources" [9]; "**Elasticity** is the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomous manner, such that at each point in time the available resources match the current demand as closely as possible" [9]; and "**Efficiency** expresses the amount of resources consumed for processing a given amount of work" [9]. We use these definitions throughout this paper to derive metrics for each quality property. In the next section, we exemplify these definitions based on our running example.

3. MOTIVATING EXAMPLE

As an example scenario, we consider a simplified online bookshop. An enterprise assigns a software architect to design this shop, given the following requirements:

R_{fct}**: Functionality** In the shop, customers shall be able to browse and order books.

R_{scale}**: Scalability** The enterprise expects an initial customer arrival rate of 100 customers per minute. It further expects that this rate will grow by 12% in the first year, i.e., increase to 112 customers per minute. In the long run, the shop shall therefore be able to handle this increased load without violating other requirements.

R_{elast}**: Elasticity** The enterprise expects that the context for the bookshop repeatedly changes over time. For example, it expects that books sell better around Christmas while they sell worse around the holiday season in summer. Therefore, the system shall proactively adapt to anticipated changes of the context, i.e., maintain a response time of 3 seconds or less as well as possible. For non-anticipated changes of the context, e.g., peak workloads, the system shall re-establish a response time of 3 seconds or less within 10 minutes once a requirement violation is detected.

R_{eff}**: Efficiency** The costs for operating the bookshop shall only increase (decrease) by \$0.01 per hour when the number of customers concurrently using the shop increases (decreases) by 1. In other words, the marginal cost of the enterprise for serving an additional customer shall be \$0.01.

Requirements R_{scale}, R_{elast}, and R_{eff} are typical reasons to operate a system in an elastic cloud computing environment [9], i.e., an environment that autonomously provisions the required amount of resources to cope with contextual changes. Thus, the software architect designs the shop as a 3-layer Software as a Service (SaaS) application operating in a rented Infrastructure as a Service (IaaS) cloud computing environment that provides replicable virtual servers (see Fig. 1). The three layers involve the typical layers of web applications: presentation, application, and data layer.

The architect designs each SaaS layer such that it can consume a higher/lower quantity of IaaS services to sustain changing workloads. Technically, he ensures that each SaaS layer can be replicated and load-balanced over additional IaaS virtual servers (scale-out) or be removed again (scale-in). Therefore, properties like scalability (R_{scale}), elasticity (R_{elast}), and efficiency (R_{eff}) of the bookshop are inherently coupled with corresponding properties of the underlying IaaS environment. The enterprise (IaaS consumer) and the IaaS provider have, thus, to agree on measurements and thresholds for these properties. Typically, consumer and provider achieve an agreement based on negotiated SLOs as exemplified by R_{scale}, R_{elast}, and R_{eff}. However, currently there is a lack of agreed-on metrics for scalability, elasticity, and efficiency in the context of cloud computing. This lack results in too few SLOs or SLOs that cannot be quantified and checked by cloud consumers. In consequence, there is the risk that requirements R_{scale}, R_{elast}, and R_{eff} cannot be fulfilled due to a non-anticipated (and contractually uncheckable) behavior of the underlying IaaS environment.

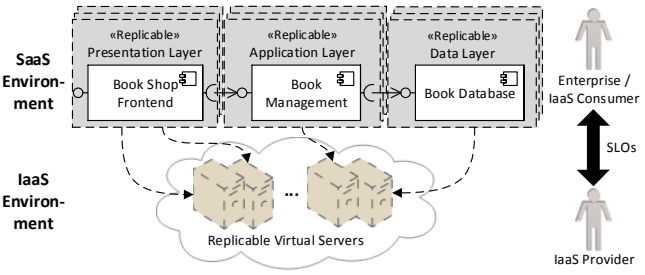

Figure 1: Overview of the simplified online bookshop.

4. DERIVING METRICS WITH GQM

We derive our metrics with the goal question metric (GQM) method [15] in a systematic top-down fashion by first defining the goal to analyze cloud computing system designs, e.g., for design analysis or SLO specification (conceptual level). Second, we formulate questions that help achieving the goal (operational level). Finally, we identify metrics that allow us to answer the questions (quantitative level).

4.1 Goal

Table 1 shows the goal in form of the GQM goal definition template. The metrics we want to define in this paper shall help to analyze (purpose) the scalability, elasticity, and efficiency (issue) of cloud computing system designs (object) from the viewpoint of a software architect (viewpoint).

Note that the derived metrics are not necessarily generalizable or applicable for different objects or viewpoints. For example, cloud computing vendors may perceive the efficiency of cloud system software different than software architects and, thus, need different metrics.

Table 1: Goal definition according to GQM plan

Purpose	Analyze
Issue	the scalability, elasticity, and efficiency of
Object	cloud computing system designs
Viewpoint	from a software architect's viewpoint

4.2 Questions

Next, we define questions that help achieving our defined goal. We define questions for each quality property separately. All questions are indicator questions for the according quality property as defined in Sec. 2.

Questions for Scalability.

According to the definition, scalability is an *ability*, i.e., a system is either scalable or it is not. Hence, we define questions whether a system is able to fulfill its requirements under increasing workload. Moreover, the increase rate of the workload can be a relevant context factor.

$Q1_{scale}$ Does the system fulfill its requirements when the workload increases (from workload WL_X to WL_Y)?

$Q2_{scale}$ Does the system fulfill its requirements when the workload increases with rate R (from workload WL_X to WL_Y within time t)?

Questions for Elasticity.

Elasticity is, according to the definition, the degree to which a system is able to autonomously adapt to workload changes. Thus, we define questions that consider the time it takes for the system to adapt.

$Q3_{elast}$ How often does the system violate its requirements under workload WL_X in time period Δt?

$Q4_{elast}$ From a point the system violates its requirements, how long does it take before the system recovers to a state in which its requirements are met again?

Questions for Efficiency.

Efficiency relates the amount of demanded resources to the amount of work requested. Hence, we formulate questions that ask for this relation.

$Q5_{eff}$ How close to the actual resource demand can the system align the resource provisioning?

$Q6_{eff}$ What is the amount of resources, autonomously provisioned by the system, for a given workload WL_X?

4.3 Metrics

In this section, we summarize general requirements for metrics and concrete requirements for cloud computing system metrics. We then derive our metrics that answer the questions from Sec. 4.2 in two steps. In the first step, we derive exemplary metrics (EM) for scalability, elasticity, and efficiency for the example system in Sec. 3. In the second step, we generalize these metrics to answer the questions for arbitrary cloud computing systems.

4.3.1 Requirements for Derived Metrics

The metrics we define have to meet four typical characteristics of metrics [7] in order to be applicable by software architects: (1) *quantifiability*, (2) *repeatability*, (3) *comparability*, and (4) *easy obtainability*. Additionally, we require our metrics to be (5) *context dependent* to reflect the context dependency of cloud computing systems. Figure 2 shows parts of the context that impact the quality of a cloud computing system as a feature diagram. This context covers the system's workload, i.e., work and load, and deployment, i.e., replication of components, processor speed, memory size, network speed, etc. This entire context is subject to change at run-time in a typical cloud computing environment. Hence, metrics need to have a context parameter to enable the comparison of implementations in different contexts. For example, system $\mathcal{S}_A$ may have less SLO violations per minute with workload WL_X (context) but more SLO violations per minute with workload WL_Y (different context) compared to system $\mathcal{S}_B$.

4.3.2 Derived Metrics from Example Scenario

In this section, we define exemplary metrics (EM) that can be used to answer the questions from Sec. 4.2. In this section, we restrict these metrics to evaluate whether the requirements for the example scenario in Sec. 3 are fulfilled. In Sec. 4.3.3, we generalize these metrics for arbitrary cloud computing systems.

The first requirement R_{fct} in our example scenario is a general requirement that defines the basic functionality of the bookshop. However, metrics for functional requirements are out of the scope of this paper.

Exemplary Scalability Metrics.

R_{scale} specifies the system's required scalability, i.e., the system's ability to make use of additional resources at increasing workloads. Hence, this requirement is dependent on the context, e.g., the load specified as arrival rates. The book enterprise has to specify arrival rates, e.g., by estimating current sales, sale trends, and seasonal sale variability.

The bookshop's software architect checks whether the designed system fulfills requirement R_{scale} by evaluating questions $Q1_{scale}$ and $Q2_{scale}$ for this design. A metric EM_{scale}, defined as the maximum workload the system can handle without violating requirement R_{scale}, answers question $Q1_{scale}$. For example, the software architect can measure this metric by predicting the performance for the bookshop design with the increased workload of 12% as expected by the book enterprise (predictions can, e.g., be conducted using queuing networks, c.f. [14]).

The enterprise does not specify at which rate the workload increases, e.g., linearly or exponentially. Therefore, to answer question $Q2_{scale}$, the bookshop's software architect needs a metric EM_{rate}, defined as the rate a system can scale up to a certain maximum workload. For example, the software architect can measure this metric by predicting the performance of the bookshop design under increasing workload, e.g., a linear increasing workload of one additional customer per month. Afterwards, the architect can discuss the quantified requirement with the enterprise.

Exemplary Elasticity Metrics.

R_{elast} specifies the bookshop's required elasticity, i.e., the degree to which the bookshop is able to adapt to workload changes by autonomously provisioning and deprovisioning cloud resources. In general, these workload changes (context) can be either anticipated or impossible to anticipate. As described in Sec. 3, the enterprise can anticipate variability of the bookshop's customers demand, i.e., the enterprise estimates to sell more books around Christmas than in mid-summer. Other short-term variations of the workload cannot be anticipated, e.g., peak workloads.

The bookshop's software architect can evaluate whether R_{elast} is fulfilled by answering questions $Q3_{elast}$ and $Q4_{elast}$ for the bookshop design. The cloud computing system can potentially cope with both, anticipated and non-anticipated, workload variability. However, considering the fact that resources can be available with delay, the system will likely violate requirement R_{elast} until the time additionally provisioned resources are available. A metric EM_{viol}, defined as

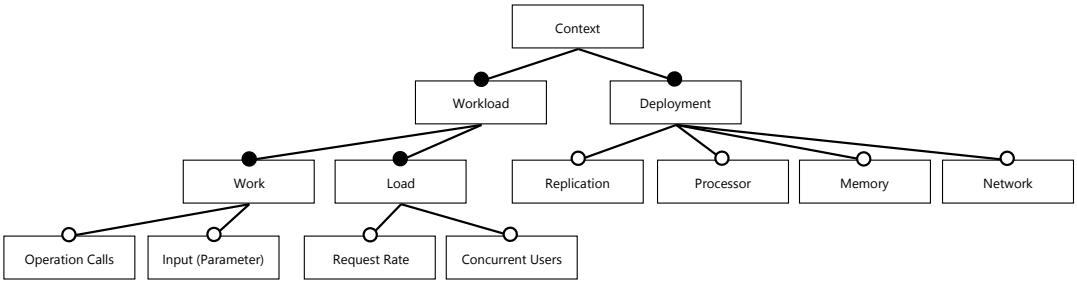

Figure 2: Feature diagram showing aspects of context.

number of requirement violations for a given workload, is a mean for the software architect to answer question $Q3_{elast}$. A metric EM_{adapt}, defined as the time between detection of a requirement violation and the time when the requirement is fulfilled again, e.g., by autonomous resource provisioning, answers question $Q4_{elast}$. For example, the bookshop's architect can measure both metrics, EM_{viol} and EM_{adapt}, by simulating the bookshop's autonomous behavior using self-adaptive queuing networks [2].

Exemplary Efficiency Metrics.

R_{eff} specifies the required efficiency of the bookshop regarding its resource consumption. The bookshop's autonomous provisioning and deprovisioning adapts the resource consumption to the bookshop's workload, i.e, the workload is important context here as well. Furthermore, the operation costs of the bookshop depend on this resource consumption as the enterprise has to pay for the cloud resources in a pay-per-use fashion.

The software architect can evaluate whether the R_{eff} is fulfilled by answering questions $Q5_{eff}$ and $Q6_{eff}$. A metric EM_{close} can be defined as the difference between the bookshop's amount of provisioned resources for a workload WL_X and the minimum amount of resources required to cope with that workload WL_X without violating the stated requirements. The bookshop's software architect can use this metric to answers question $Q5_{eff}$. A metric EM_{prov}, defined as the amount of resources provisioned for a workload WL_X, answers question $Q6_{eff}$. Whereas the marginal costs can be calculated directly from the metric EM_{prov}, EM_{close} supports the software architect to determine whether the required marginal costs are achievable. Again, both metrics can be measured by simulating the bookshop's autonomous behavior under variable workload, for example.

4.3.3 Generalized Metrics

We define initial new metrics that we derived from the six exemplary metrics (EM) from the previous subsection. For each metric, we define the corresponding quality property it quantifies, what is being measured, on which contextual properties the measurement depends, and the scope and unit of the measurement result. To guarantee objectivity, our metrics rely only on externally observable properties of a system, e.g., costs per time. Reproducibility is guaranteed by specifying the contextual properties on which the metric depends. Thus, context dependency is also guaranteed. By defining the measurement result's scope as ordinal numbers, we guarantee that the metric reflects a testable quantification of the quality property. Additionally, we provide a small example for each metric. Table 2 summarizes the following generalized metrics.

Scalability Metrics.

Scalability Range (ScR) Based on EM_{scale}, we define scaling range as a scalability metric that reflects a cloud computing system's ability to achieve its SLOs in a certain workload range, e.g., a range of request rates. For each single workload within this range, the system achieves its SLOs. The workload range is defined as a maximum workload. For example, a perfectly scalable system $\mathcal{S}_A$ can scale up to a infinite request rate, i.e., $ScR_{\mathcal{S}_A} = \infty\ req/min$. A less scalable system, for example, scales up to a request rate of 112 requests per minute, i.e., $ScR_{\mathcal{S}_B} = 112\ req/min$.

Scalability Speed (ScS) Based on EM_{rate}, we define scalability speed ScS as a workload range with a maximal change rate in which a system can scale. This metric is a scalability metric which additionally considers the rate at which a system can scale. That is, the metric defines that a system is able to achieve its SLOs at each time when the workload changes at a maximal changing rate. The rate is defined by a maximum workload and an increase rate. For example, a scalable system can scale up to a request rate of 112 req/min with a linear increase rate of 1 additional requests per month, i.e., $ScS_{\mathcal{S}_A} = (112\,req/min, 1\,req/month)$.

Elasticity Metrics.

Mean Time To Quality Repair ($MTTQR$) We derive the mean time to quality repair metric from EM_{adapt}, the measure how quickly a system can adapt to workload changes. It is a measure for elasticity and depends on a workload delta, i.e., the increase/decrease between two workloads. $MTTQR$ defines the mean time a system needs to reestablish its SLOs when the workload increases/decreases for a defined workload delta specified as factor (real number). Hence, $MTTQR$ is measured in time units. Since it defines a mean time, $MTTQR$ is specific for a specified time frame in which the mean is calculated. For example, with the same workload increase factor of 1.2, a perfectly elastic system will adapt itself to the increasing workload within zero time, i.e., $MTTQR_{\mathcal{S}_A,1d}(1.2\ req/s) = 0\ min$. A less elastic system, e.g., will need a mean time of 10 min (calculated over one day) to adapt itself to increasing workload after it detects the workload increase, i.e., $MTTQR_{\mathcal{S}_B,1d}(1.2\ req/s) = 10\ min$.

Number of SLO violations ($NSLOV$) The number of SLO violations in a defined time interval is derived from EM_{viol}. This metric measures elasticity of a system. The workload delta is specified as a factor (real number) as well.

Table 2: Derived quality property metrics for cloud computing systems

Metric	Unit	Example
Scalability Metrics		
Scalability Range (ScR)	max	The system scales up to 112 req./min
Scalability Speed (ScS)	(max, rate)	The system scales up to 112 req./min with linear increase rate 1 req./month
Elasticity Metrics		
Number of SLO Violations ($NSLOV$)	1/[time unit]	42 SLO (response time) violations in 1 hour
Mean Time To Quality Repair ($MTTQR$)	[time unit]	30 seconds for an additional 10 requests/hour
Efficiency Metrics		
Resource Provisioning Efficiency (RPE)	$[0, \infty]$	10% more resources than actual resource demand
Marginal Cost (MC)	[monetary unit]	\$1.00 for an additional 100 requests/hour

$NSLOV$ reflects how often a system violates its SLOs when workload changes at a given rate, measured as a real number. For example, with a workload increase factor of 1.2, a perfectly elastic system would have 0 SLO violations per request, i.e., $NSLOV_{\mathcal{S}_A}(1.2\ req/s) = 0$. In contrast, a non-elastic system will violate its SLO for each request, i.e., $NSLOV_{\mathcal{S}_A}(1.2\ req/s) = 1$.

Efficiency Metrics.

Resource Provisioning Efficiency (RPE) We define resource provisioning efficiency (RPE) as a metric to measure a system's efficiency in a specified workload delta based on EM_{close}. That is, the metric measures the mismatch between actual resource utilization and resource demand while the workload is changing. We measure this mismatch in percentage, i.e., as a real number. A perfectly efficient system will adapt its resource demand exactly to the resource demand at all times. For example, if the workload increases with factor 1.2 the system will provision exactly that amount of additional resources required to cope with this additional workload, i.e., $RPE_{\mathcal{S}_A}(1.2\ req/s) = 0$.

Marginal costs (MC) We derive the *marginal costs* for a specified workload delta from EM_{prov}. Marginal costs are the operation costs to serve one additional workload unit, thus, measuring the efficiency of a cloud computing system. For example, the operation costs to serve 20% additional requests per second (factor 1.2) can be \$1.00 for system $\mathcal{S}_A$, i.e., $MC_{\mathcal{S}_A}(1.2) = \$1.00$.

5. RELATED WORK

In this section, we present related work that also targets metrics for scalability, elasticity, and efficiency in the context of cloud computing and SLO specification. Where applicable, we classify found metrics using our GQM plan in Sec. 6.

In the area of **scalability metrics**, Bondi [5] further divides scalability into structural scalability ("ability to expand in a chosen dimension without major modification") and load scalability ("ability of a system to perform gracefully as the offered traffic increases"). In terms of this classification, we focus on load scalability in our work. However, in contrast to Bondi, we also provide concrete metrics for load scalability and apply these to the cloud computing context. Duboc et al. [6] formally define scalability requirements and provide a method to analyze a software model for scalability obstacles. A scalability obstacle could also be defined and detected using our metrics. In contrast to the approach from Duboc et al., our metrics are closer to SLOs in their current form. Jogalekar and Woodside [11] present a scalability metric for general distributed systems. Their scalability metric also includes an efficiency measure. In our work, we distinguish between scalability and efficiency as two different metrics. Furthermore, in contrast to Jogalekar and Woodside, our metrics are focused on cloud computing environments with their particular characteristics.

Herbst et al. [9] provide a set of **elasticity metrics** based on speed and precision (w.r.t. avoiding under- and overprovisioning) of scale-in and -out. Because their goal is a benchmarking methodology for elasticity, they can assume full knowledge about the resource usage of the benchmarked application. However, in our case, we assume that this knowledge is unavailable because details on resources are implementation decisions. In contrast, we consider requirements specified between cloud consumer and provider. This lack of knowledge necessarily leads to different metrics as by Herbst et al., e.g., considering SLO violations instead of resource usage transparent to consumers. Folkers et al. [8] and Islam et al. [10] both provide elasticity metrics that meet this requirement regarding knowledge. However, they lack the distinction between elasticity and efficiency because they both use cost metrics for elasticity, thus, eliminating the possibility to investigate both properties in separation.

Roloff et al. [12] define basic **efficiency metrics** for high performance computing in cloud computing environments. They define cost efficiency as the product of costs per hour and average performance. In contrast to our work, they neglect the context, e.g., actual workload, and only take the average performance. Berl et al. [3] address energy efficiency in all technical components of cloud computing, e.g., servers, networks as well as network protocols. We do not address energy efficiency directly but only resource provisioning efficiency in this paper. Investigating whether efficient provisioning of resources positively correlates with energy efficiency is left as future work.

These related works focus on single quality properties. In contrast, the Cloud Services Measurement Initiative Consortium (CSMIC) provides a standard **measurement framework**, called the Service Measurement Index (SMI), that covers all quality properties considered important for cloud computing [13]. CSMIC particularly provides metrics for these quality properties, intended to be used by cloud consumers and cloud providers. Their framework allows for a structured classification of quality properties, similar to our GQM plan. However, CSMIC does not address the deviation of their suggested metrics, thus, leaving open what limitations come with their metrics and whether other metrics are feasible as well.

6. CLASSIFICATION OF METRICS IN RELATED WORK

In this section, we classify metrics identified in related work (Sec. 5) by relating these metrics to the questions of our GQM plan. In doing so, we show that we can systematically make assumptions and limitations of existing metrics explicit. Note that we classify only a few related metrics, for illustration.

Scalability metric by Jogalekar and Woodside [11] Jogalekar and Woodside provide a scalability metric that can directly be used to answer $Q1_{scale}$ ("Does the system fulfill its requirements when the workload increases (from workload WL_X to WL_Y)?"). They use workload as the main input context factor to their metric, thus, complying to our scalability question. As an output, they determine a productivity factor that states whether system requirements can be fulfilled (using a threshold for this factor). Also this idea complies to our question. However, to calculate the productivity factor, they require knowledge about the operation costs for a given workload. In contrast, we do not limit our scalability metrics to this knowledge about costs and require it for elasticity metrics only.

Elasticity and efficiency metrics by Herbst et al. [9] Herbst et al. provide metrics that consider speed and precision (w.r.t. avoiding under- and overprovisioning) of scale-in and -out. Their ideas on speed can be used to answer our elasticity question $Q4_{elast}$ ("From a point the system violates its requirements, how long does it take before the system recovers to a state in which its requirements are met again?"). However, to detect requirement violations, they assume to have already an implemented system at hand; design-time (e.g., simulation-based) approaches do not work with their metric. Their ideas on precision fit to our elasticity question $Q5_{eff}$ ("How close to the actual resource demand can the system align the resource provisioning?"). Again, their metric requires an implemented system to determine the actual resource usage.

7. CONCLUSION

In this short research paper, we argue for the need of novel metrics for quality properties of cloud computing systems. Using the GQM method, we systematically derive an initial set of six metrics for scalability, elasticity, and efficiency. Moreover, by using our GQM plan, we classify existing metrics to make their limitations explicit.

Our systematically derived metrics help software architects, requirements engineers, testers, etc. to design and analyze cloud computing systems. Our GQM plan helps them to consider limitations of such metrics during these tasks.

Future work is directed towards the development of analysis methods and tools that enable software architects to verify the fulfillment of scalability, elasticity, and efficiency requirements of their cloud computing applications already at design time. Understanding limitations of metrics is essential for this purpose.

8. ACKNOWLEDGMENTS

This work is supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901). The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant no 317704 (CloudScale).

9. REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.

[2] M. Becker, S. Becker, and J. Meyer. SimuLizar: Design-time modelling and performance analysis of self-adaptive systems. In *Proceedings of Software Engineering 2013 (SE2013), Aachen*, 2013.

[3] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis. Energy-efficient cloud computing. *The Computer Journal*, 53(7):1045–1051, 2010.

[4] G. Bolch, S. Greiner, K. S. Trivedi, and H. de Meer. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation With Computer Science Applications*. 1998.

[5] A. B. Bondi. Characteristics of scalability and their impact on performance. In *WOSP '00*, pages 195–203, New York, NY, USA, 2000. ACM.

[6] L. Duboc, E. Letier, and D. S. Rosenblum. Systematic elaboration of scalability requirements through goal-obstacle analysis. *IEEE Transactions on Software Engineering*, 39(1):119–140, Jan. 2013.

[7] T. Erl, Z. Mahmood, and R. Puttini. *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall, 2013.

[8] E. Folkerts, A. Alexandrov, K. Sachs, A. Iosup, V. Markl, and C. Tosun. Benchmarking in the cloud: What it should, can, and cannot be. In *TPCTC*, pages 173–188, 2012.

[9] N. R. Herbst, S. Kounev, and R. Reussner. Elasticity: What it is, and What it is Not. In *ICAC '13*, 2013.

[10] S. Islam, K. Lee, A. Fekete, and A. Liu. How a consumer can measure elasticity for cloud platforms. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pages 85–96, New York, NY, USA, 2012. ACM.

[11] P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *IEEE Trans. Parallel Distrib. Syst.*, 11(6):589–603, June 2000.

[12] E. Roloff, M. Diener, A. Carissimi, and P. Navaux. High performance computing in the cloud: Deployment, performance and cost efficiency. In *CloudCom '12*, pages 371–378, 2012.

[13] J. Siegel and J. Perdue. Cloud services measures for global use: The service measurement index (smi). In *SRII Global Conference (SRII), 2012 Annual*, pages 411–415, July 2012.

[14] C. U. Smith. *Performance engineering of software systems*. Software Engineering Institute series in software engineering. Addison-Wesley, 1990.

[15] R. van Solingen, V. Basili, G. Caldiera, and H. D. Rombach. *Goal Question Metric (GQM) Approach*. John Wiley & Sons, Inc., 2002.