Proceedings of the Second HPI Cloud Symposium "Operating the Cloud" 2014

Sascha Bosse, Mohamed Esam Elsaid, Frank Feinbube, Hendrik Müller (Eds.)

Technische Berichte Nr. 94

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

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

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 94

Sascha Bosse | Mohamed Esam Elsaid | Frank Feinbube | Hendrik Müller (Eds.)

Proceedings of the Second HPI Cloud Symposium "Operating the Cloud" 2014

Universitätsverlag Potsdam

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

Universitätsverlag Potsdam 2015
http://verlag.ub.uni-potsdam.de/

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

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

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

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URN urn:nbn:de:kobv:517-opus4-76654
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-76654

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-319-0

Preface

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

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

In this workshop proceedings the results of the second HPI cloud symposium ”Operating the Cloud” 2014 are published. We thank the authors for exciting presentations and insights into their current work and research. Moreover, we look forward to more interesting submissions for the upcoming symposium in 2015.

Contents

Evaluating IT Service Design Alternatives With Respect to Availability, Response Times and Costs

Sascha Bosse, Johannes Hintsch, Christian Schulz, Matthias Splieth,
Hendrik Müller, and Klaus Turowski

Very Large Business Applications Lab
Otto von Guericke University Magdeburg, P.O. Box 4120, Magdeburg
{sascha.bosse|johannes.hintsch|christian.schulz|matthias.splieth
hendrik.mueller|klaus.turowski}@ovgu.de

The paradigm of cloud computing has become an important driver for IT service-orientation. A major challenge for IT service providers is to satisfy service-level objectives in terms of availability and performance with respect to IT system landscape costs. Decisions affecting these aspects are mainly made in the service design stage in which reliable quantitative data is often not available. Therefore, analytical prediction models using architectural information are recommended. In this paper, an analytical prediction model based on Petri net simulation is conceptualized to support decision makers by comparing design alternatives with respect to availability, performance and costs. The developed concept could be evaluated in a real-world case study and presents a step towards a scalable decision-support framework for IT service management in the service design stage.

1 Introduction

Since the advent of cloud computing, it has become a very popular paradigm to obtain computing resources on-demand. Such cloud services are elastic IT services which are provisioned using pooled resources and are monitored continuously to ensure cloud service quality [16]. This quality is measured by IT service metrics such as availability or response time, for which guarantees are given in service-level agreements (SLA) by the IT service provider.

One challenge that arises, especially for cloud service providers, is the need to evolve their IT system landscapes cost-efficiently in order to consolidate computing capacities and to face new technologies or business demands [7]. On the other hand, service-level objectives – for instance for the service availability – should not be violated in this evolution, since this could lead to penalty costs or loss of reputation for the IT service provider [8]. Hence, a provider needs to evaluate IT system landscape design alternatives with respect to IT service quality and costs. Since IT system landscapes, especially those of cloud service providers, are often complex and heterogeneous, this is not a trivial problem [26].

A possible way to analyze IT service quality is the application of performability modeling techniques in which performance as well as dependability aspects are investigated [10]. The IT Infrastructure Library (ITIL), a good-practice set for IT service management, recommends the application of these analytical prediction

models in the service design stage [11] since decisions affecting IT service quality are mainly made in this stage and are costly to correct in the subsequent stages [26]. However, existing approaches for perfomability modeling of IT services concentrate on single non-functional metrics, ignoring other metrics as well as the inter-dependencies between them [17]. Therefore, a suitable and scalable method for performability modeling of IT services is conceptualized in this paper. It is designed to predict IT service availability and response times as two of the most crucial performability metrics [13] as well as the IT system landscape costs. Hence, this concept is supposed to be a step towards effective decision-support when evolving IT system landscapes in the service design stage.

Before the developed concept is presented in section 3, the current state of the art in performability modeling for IT services is discussed in section 2. The implemented prototype is then used to conduct experiments of a real-world case study in order to demonstrate the ability of the developed concept to support decision-making. These evaluation results are presented in section 4. Section 5 concludes the contribution by discussing the results and providing an outlook on further research activities.

2 State of the Art

This section presents the current state of the art for predicting availability, performance and costs of an IT service.

2.1 Availability Prediction

The availability of an IT service is defined by the ITIL as "the ability of a service [...] to perform its agreed function when required" [11]. Analytical approaches for predicting IT service availability are applicable in the design stage since the system is modeled as a composition of components for which availability can be estimated. Depending on the technique with which the service availability is computed from the availability of components, analytical prediction models can be divided into combinatorial, state-space and hierarchical methods [27].

Combinatorial approaches assume that the the modeled components are mutually independent. Hence, service availability can be easily computed by using the probability theory. Components in these models can be arranged serially (AND) or parallel (OR) to define their impact on service availability. Parallel connections represent redundancy mechanisms while serial connections represent critical paths in the service provisioning [28]. However, other inter-component dependencies such as maintenance or standby redundancy mechanisms cannot be modeled in combinatorial approaches, leading to a significantly limited suitability for IT service availability prediction [3].

This disadvantage is compensated in state-space-based approaches by modeling all possible states of the system landscape and the transitions between them. The availability of a service is then associated with certain states and can be computed

from the probability of occurrence of these states. Markov approaches are a very popular subcategory in these approaches since the assumption of the Markov property simplifies the availability computation, although it is unrealistic especially for component recovery times [5]. Nevertheless, modeling every possible state of the system leads to the problem of state-space explosion for real-world cases with hundreds of components, restricting the scalability of explicit state-space approaches [24]. In order to overcome this problem, different propositions such as encoding the state-space in a Petri net and Monte Carlo simulation of state-space models (e.g. in [29]) are suggested in the scientific discussion. Hierarchical approaches, combinations of system-level combinatorial and component-level state-space models, can also be used to limit model complexity [27].

2.2 Performance Prediction

As a measure of an IT service's performance, the services response time is defined as the time span between service request and response [1]. Other measures that are related to this metric are utilization and throughput [14]. The response time of an IT service is mainly determined by the network delay as well as the queuing and processing time [23]. Prototyping and benchmarking can be applied in order to analyze response times [15]. However, these methods require measured data and are, hence, not applicable in the service design stage [1]. Therefore, analytical prediction models should be applied in this stage [26].

Analytical prediction models for performance can be classified into probabilistic methods as well as state-space models based on queuing networks, Petri nets or process algebra [1]. While queuing networks are the preferred class of prediction models in this area, simulation methods are recommended for model solving since they allow for dynamic analyses [12]. The Palladio Component Model (PCM) is a prominent example for a performance prediction meta-model. It allows for the dynamic analysis of heterogeneous, component-based software systems by combining state-space models with simulation [22]. The PCM consists of four submodels, namely the component, the composition, the deployment and the usage model. However, models for component availability and hierarchical composition structures are not included in the meta model.

2.3 Costs of an IT System Landscape

Cost calculations for IT system landscapes and data centers are necessary activities in the financial management of IT services with many specialized approaches (cf. e. g. [4]). Since the scope of this paper is not focused on these calculation methods, costs are modeled on a high-level. In [2], the authors identify two basic cost types for data centers: capital expenses which can be depreciated over a certain time frame and operational expenses. Capital expenses incorporate costs for data center space and acquisition costs for components while power and maintenance costs refer to operational expenses [19]. The main driver for power costs in data centers are IT components and cooling equipment [2]. However, the power costs for cooling are directly proportional to the power costs for IT components [19]. Especially

for server components in the data center, the power consumption depends on the component's current mode (e.g. active or hot-/cold-standby) and its resource utilization [2].

3 Prediction Model

The previous section revealed that none of the identified approaches from literature is able to predict IT service availability, performance and costs in a single model with respect to their dependencies. Nevertheless, these dependencies are crucial for the accuracy of the prediction model. For instance, the performance of a service depends on the amount of available resources which again depends on the availability of IT components. On the other hand, poor performance may lead to perceived unavailability for a service consumer. However, introducing additional components for more resources or redundancy mechanisms will increase the costs of the IT system landscape.

In order to include these dependencies in a prediction model, a novel approach is developed and presented in this section. A simulation-based approach was chosen for dynamic analysis in order to quantify not only the mean value of these aspects, but also the variance. On that basis, the probability of SLA violations can be assessed more easily [9]. Since Petri nets were identified as a suitable meta model for both availability and performance prediction, the concept is developed based on colored generalized stochastic Petri nets (GSPN) (cf. e.g. [6]). Solving these models through simulation provides a high scalability since single replications can be parallelized [24].

In the following, first, a meta model for IT service availability, performance and cost prediction is introduced, before the behavior of the simulation model is presented.

3.1 Meta-Model

In order to predict availability, performance and costs of an IT service, the following relevant entities and attributes have to be modeled:

- The usage of an **IT service** is determined by a random distribution for request arrival. A timeout can be defined for each service. It determines the time after which the processing of a request is aborted and the service is perceived as unavailable by the consumer.
- For each IT service, an **operations graph** has to be defined. It arranges operations sequentially, exclusively (with path probabilities) and parallel. Hence, it corresponds to trace diagrams such as UML activity diagrams. If a request is processed through one complete path of the operations graph, the request can be responded. The time span between request arrival and response is stored as the request's response time. An exemplary operations graph is presented in figure 2.

- An **operation** is an atomic entity for request processing. Its execution requires a certain amount of resources that is provided by an associated component system. The execution time of an operation can be modeled depending on its allocated resources with random distributions.
- A **component system** is an arrangement of components that provides the components' resources to operations and models basic dependencies. Depending on the states of the involved components, the provided amount of resources can vary. Examples for possible component systems are serial, parallel redundant and voting systems.
- **Components** represent the elements of an IT system landscape that are crucial to IT service provisioning. These can be infrastructure, hardware or software components. To each component, set-up costs (are incurred at the beginning of the simulation) and power costs are assigned. The power costs for a timestep can depend on the current mode as well as on the resource utilization of this component. A component provides certain resources which can be unavailable due to failures. Different failure types can be defined for a single component determining random distributions for the time to failure and the time to recover as well as if the recovery requires operator interaction. Recovery costs can additionally be defined for each failure type, which entail when recovery happens.
- A pool of **operators** is defined for an IT system landscape. They process error-prone tasks with priorities which require manual interaction. If an interaction is imperfect, it has to be repeated. Tasks can be assigned to components or to dependencies. For each operator, a wage is defined.
- If components have inter-dependencies not covered by the component system, a **dependency** has to be defined between these components. A dependency is characterized by a trigger and an effect. For instance, a standby-dependency can be defined where the secondary component is brought online only if the primary component fails. The time between the trigger condition and the effect can be defined by a random distribution.

The availability of an IT service can then be computed by equation 1 from the set of requests reqs and the set of responses resps, the mean response time by equation 2. The total costs can be calculated by adding the capital expenses set-up and recovery costs with the operational expenses human resource and power costs. Cooling and space costs are omitted in this version of the concept, however, cooling costs can be estimated from component power costs [19].

$$A = |\mathrm{resps}|/|\mathrm{reqs}| \tag{1}$$

$$\overline{\mathrm{rt}} = \frac{1}{|\mathrm{resps}|} \sum_{\mathrm{req} \in \mathrm{resps}} \mathrm{rt}(\mathrm{req}) \tag{2}$$

3.2 Simulation Model

When a specific scenario is modeled in the meta model, a colored GSPN is generated automatically from this information. It is designed as a hierarchical model combining submodels for component behavior, operations graph and operator interaction. The request arrival distribution determines the firing rate of the operations graph's source transition which creates tokens representing a request. This token is processed through the operations graph for sequential and parallel paths (cf. figure 2). In case of exclusive paths, a random choice is made according to defined path probabilities. An operation transition can be fired if the corresponding component system provides enough resources. If the resources are available, the component systems allocates them. If this is not the case, the operation is queued until resources are available. Due to component failures, allocated resources can become unavailable, causing the abortion of the request. Abortion also happens if the time a request is processed in the system landscape is greater than the defined timeout.

The behavior of components is determined by the component model. Figure 1 presents an exemplary component model. For each resource type, a place is created with a number of tokens according to the amount of available resource entities. The places *Offline* and *Online* determine the current state of a component, while each failure type is represented by a colored token leading to different firing times for the failure and recovery transition. If a failure occurs, the component's resources are blocked immediately for operations. When the recovery process is finished, the resources are made available again. If manual interaction is required for recovery, the transition can only fire if an operator token is assigned to this component from the operator pool.

By the start of a simulation run, all components are set online and set-up costs are stored. By the end of a simulation run, availability, mean response time and costs of the service are calculated. Since one simulation run represents only one possible system behavior, a sufficient number of runs has to be executed until statistically significant results can be computed using confidence intervals.

4 Case Study Evaluation

In order to demonstrate the correctness and the feasibility of the proposed prediction model, experiments were conducted with a prototype implementation of the concept using the *java*-based simulation framework *AnyLogic 6.9.0*. The verification of the prototype was performed by comparison of results with other analytical models as well as by sensitivity tests where no comparison was possible. After the correctness of the prototype was verified, a real-world case study was conducted by analyzing a website delivery service. Therefore, first this service was modeled using the meta model from section 3 in order to create a Petri net simulation model.

In figure 2, the operations graph of the service is displayed. In ninety percent of the cases, the requested HTML page has already been rendered before and

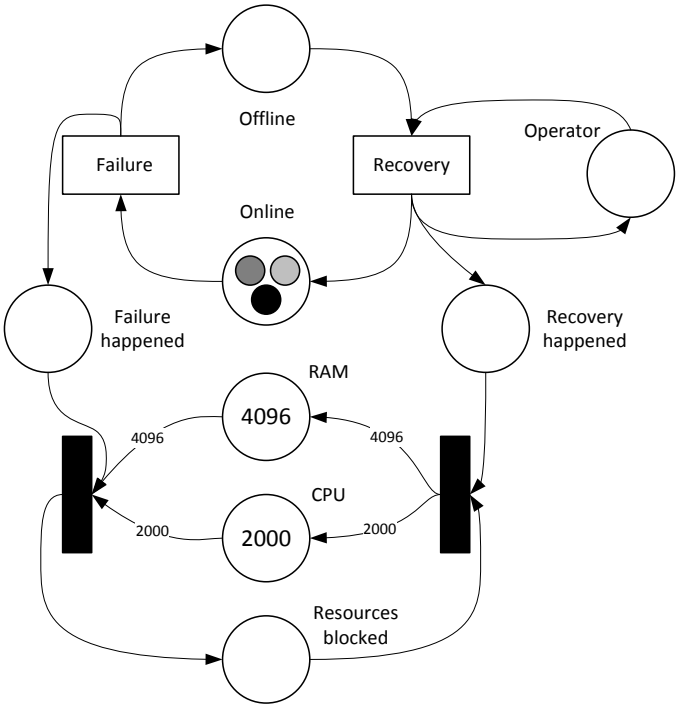

Figure 1: Exemplary Component Petri Net Model

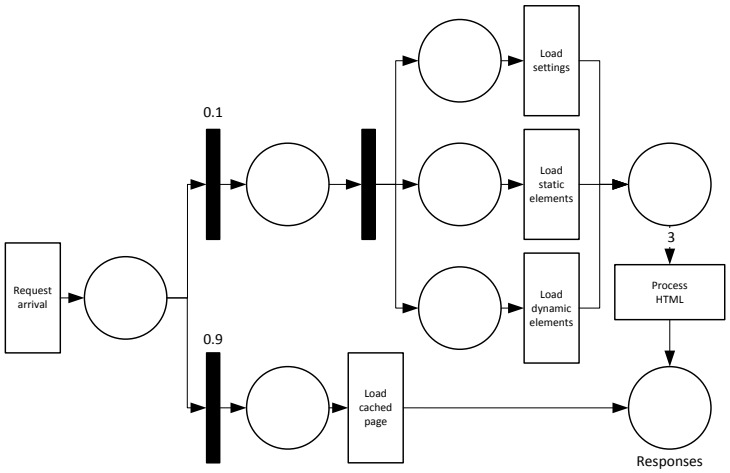

Figure 2: Petri Net Operations Graph for the Website Delivery Service

is provided by a *memcached 1.4.2* memory caching system. If the page has not been rendered before, it has to be created according to the information from a

TYPO3 4.7.6 content management system running on an *Apache 2.2.14* web server. Therefore, settings, static elements and dynamic elements are rendered parallel using data from a *MySQL 5.1.72* database operated on an *Ubuntu 10.04 LTS* server. After these templates have been rendered, the HTML page can be created and delivered to the user. Each operation requires 256 MiB of main memory and its execution time depends on the amount of allocated processing power. For 250 MHz of processing power, the execution time of the operations were measured and defined as triangular distributions according to table 1. An uncorrectable failure was assigned to each component, while the failure and recovery times were modeled as exponentially distributed random variables with mean values derived from literature according to table 2. In the original configuration, all software components are hosted on a single server with an *Intel(R) Xeon(R) X5650*@2.67 GHz and 4096 MiB of RAM.

Table 1: Execution Time Distributions for Operations of the Website Delivery Service

	Execution time triangular distributions in ms		
Operation	Minimum	Mode	Maximum
Load cached page	855	957	1,249
Load settings	16	31	118
Load static content	56	210	1,117
Load dynamic content	1,010	1,719	2,255
Process HTML page	69	74	87

Table 2: Mean Time to Failure (MTTF) and Recover (MTTR) for Case Study Components

Component	MTTF in h	MTTR in h
Web server/caching [17]	8,760	1.44
Database server [17]	8,760	1.73
RAM DIMM [25]	38,672	1.45
Physical server [21]	5,463	1.97

In order to compare the simulation results to real values, the website delivery service was monitored to parametrize the model. It was instantiated with a request arrival rate of 0.12276 requests per minute and a timeout of 10 seconds. Hence, the mean response time from reality (0.99894 seconds) could be compared to the mean

value of the expected response time after 100 replications simulating one year of operation (1.142 seconds). This comparison results in a relative difference of 14.32 %, which is within the tolerable difference of 30 % for response time prediction defined in literature [14]. Therefore, the validity of the prediction model for response times cannot be denied.

After the validation experiment, the request arrival rate is varied in order to analyze availability, response time and costs in a stress test. This is done for the original configuration as well as for two hypothetical scenarios: in these scenarios, a second server is installed with 4×3.0 GHz processing power and 8 GiB of RAM. In the parallel configuration, both servers are hosted parallel redundant while in the standby configuration, the old server is run in standby mode, only to be activated if the new server fails. For all configurations, the service-level objective (SLO) for the service availability is set to 99.5 % and for the service mean response time to 1.2 seconds. In order to compute power costs, data from the *SPECpower_ssj2008* benchmark for server systems was used [2], the power consumption in standby mode is set to 10 % of the active idle value [18]. The energy price is set to 0.2306 € per kWh[1].

In figure 3, the results of the stress test are displayed for the original configuration for request arrival rates between 0.12276 and 1 request per minute. It can be analyzed that the SLO of 99.5 % availability is violated if the arrival rate exceeds 0.7 requests per minute. This analysis was conducted for the other two configurations as well, the results are presented in table 3. In order to compare the different configurations, the table displays the mean results for the original request arrival rate and the mean results for the last arrival rate where the defined SLO is not violated (grey highlighted).

It is obvious that the introduction of redundancy mechanisms in the two hypothetical scenarios considerably improves the resilience of the service in terms of request arrivals. While the original configuration can only handle 0.7 requests per minute before the SLO is violated, the standby configuration can handle 159, the parallel configuration even 216 requests per minute before the mean response time exceeds 1.2 seconds. However, it can be stated that in the standby configuration, significantly less power costs are generated. Therefore, the standby configuration would be the better alternative if the request arrival rate does not exceed 159 requests per minute. Using this information, mechanisms could be implemented that automatically switch the service to a parallel configuration if the request arrival rate requires that in order to hold the SLO.

[1] *E.ON ProfiStrom* tariff at 11-01-2013 `https://www.eon.de/de/angebot/gk/produkteUndPreise/Gewerbekunden/Strom/E.ON_ProfiStrom_ET/index.htm`

Table 3: Simulation Results for the Three Configurations of the Case Study

Configuration	Request arrival rate per min	Mean availability in %	Mean response time in s	Mean power costs in €
Original	0.1226	99.87100	1.142	349.82
	0.7	99.52393	1.146	350.85
Standby	0.1226	99.90857	0.988	384.05
	159.0	99.58274	1.198	448.49
Parallel	0.1226	99.99994	0.988	693.01
	216.0	99.79986	1.199	774.02
SLO		99.50000	1.200	

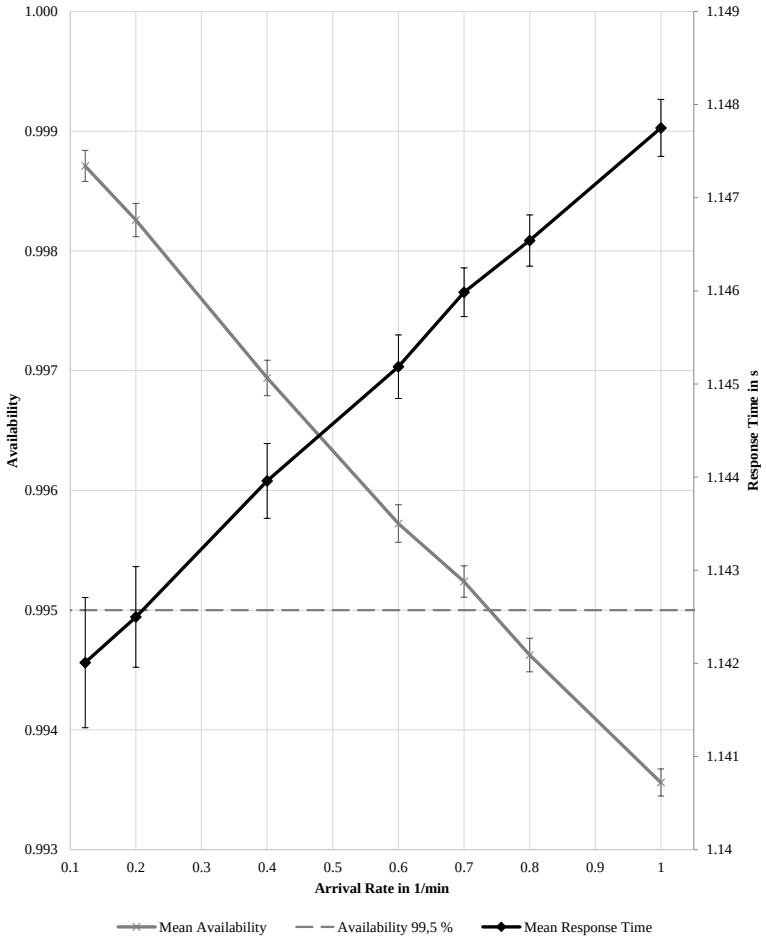

Figure 3: Availability (Primary Axis) and Response Time (Secondary Axis) Confidence Intervals for Different Request Arrival Rates in the Deliver Website Case Study

5 Conclusion

In this paper, a concept for predicting availability, response time and costs of IT services was developed. It aims at supporting decision makers in the service design stage to evaluate design alternatives in terms of performability service-level objectives and costs. Therefore, a Petri net-based simulation model was designed that incorporates the dependencies between the output variables in a single model. For instance, response times and power costs depend on the current resource utilization in the data center. The resource capacities, on the other hand, can be affected by component unavailability and poor response times will also lead to perceived unavailability.

The analysis of the state of the art in section 2 revealed that no existing approach for performability evaluation is able to model these dependencies in a single model. In section 3, the meta model of the prediction framework presents the modeled entities and attributes, before the Petri net simulation model is introduced in the second part of this section. A demonstration of the concept's abilities is presented in the evaluation section 4. Therefore, a case study of a website delivery service was analyzed. The conducted experiments indicate that the developed concept is feasible to support decision making in the service design stage by comparing different design alternatives with respect to availability, performance and costs.

Although the concept includes features such as trace-based performance analysis, component availability, error-prone operator interaction and generic inter-component dependencies, it has to be extended in the future in order to achieve sufficient accuracy in the field of performability analysis. For instance, maintenance activities in order to decrease component failure rate as well as costs of space and cooling have to be integrated into the model. Also architectural dependencies between software components could be predefined to increase the usability of the concept.

Future research in the area of performability analysis in general has to consider two major problems: model creation and parametrization. The model creation process is often done manually and is, therefore, error-prone. In addition to that, in the majority of the performability prediction approaches, models have to be created in (semi-)formal modeling languages which can be evaluated directly. This process requires knowledge which is often not available for decision makers in the service design stage. Therefore, this contribution addresses this problem by providing a meta modeling language which is automatically translated to Petri nets. In future work, this could be supported by information from existing sources such as configuration management databases which was discussed e.g. in [17].

Future research in the area of performability analysis in general has to consider two major problems: model creation and parametrization. The model creation process is often done manually and is, therefore, error-prone. In addition to that, in the majority of the performability prediction approaches, models have to be created in (semi-)formal modeling languages which can be evaluated directly. This process requires theoretical knowledge which is often not available for decision makers in the service design stage. Therefore, this contribution addresses this problem by

providing an meta modeling language which is automatically translated to Petri nets. In future work, this could be supported by information from existing sources such as configuration management databases which was discussed e.g. in [17].

Another important problem of analytical prediction models is the model parametrization. For hardware components, for example, manufacturers' data is often over-optimistic [20] and software component parameters can only be estimated in early software development phases since benchmarks cannot be used in these phases. For this problem, one approach could be to find an adequate abstraction level where previously measured data, for instance from similar services, can be incorporated into analytical models.

The consideration of the mentioned aspects in future work should improve the usability and accuracy of performability analysis models applicable in the service design stage and, therefore, supports IT service managers to trade-off performability service-level objectives and costs.

References

[1] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. "Model-Based Performance Prediction in Software Development: A Survey". In: *IEEE Transactions on Software Engineering* 30 (2004), pages 295–310.

[2] L. Barroso, J. Clidaras, and U. Hölzle. *The Datacenter as a Computer*. Edited by M. Hill. 2nd edition. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2013.

[3] G. Callou, P. Maciel, D. Tutsch, J. Araújo, J. Ferreira, and R. Souza. "A Petri Net-Based Approach to the Quantification of Data Center Dependability". In: *Petri Nets – Manufacturing and Computer Science*. Edited by P. Pawlewski. InTech, 2012. Chapter 14, pages 313–336.

[4] D. Cannon. *ITIL Service Strategy 2011 Edition*. The Stationery Office, 2011.

[5] C. Chellappan and G. Vijayalakshmi. "Dependability modeling and analysis of hybrid redundancy systems". In: *International Journal of Quality & Reliability Management* 26 (2009), pages 76–96.

[6] G. Ciardo, J. Muppala, and K. Trivedi. "SPNP: Stochastic Petri Net Package". In: *Proceedings of the 3rd International Workshop PNPM*. IEEE Computer Society, 1989, pages 142–151.

[7] J. Eckert, N. Repp, S. Schulte, R. Berbner, and R. Steinmetz. "An Approach for Capacity Planning of Web Service Workflows". In: *Proceedings of the 13th Americas Conference on Information Systems (AMCIS)*. 2007.

[8] V. C. Emeakaroha, M. A. S. Netto, R. N. Calheiros, I. Brandic, R. Buyya, and C. A. F. D. Rose. "Towards autonomic detection of SLA violations in Cloud infrastructures". In: *Future Generation Computer Systems* 28.7 (2012), pages 1017–1029.

[9] U. Franke. "Optimal IT Service Availability: Shorter Outages, or Fewer?" In: *IEEE Transactions on Network and Service Management* 9 (2012), pages 22–33.

[10] B. R. Haverkort and I. Niemegeers. "Performability modelling tools and techniques". In: *Performance Evaluation* 25 (1996), pages 17–40.

[11] L. Hunnebeck. *ITIL Service Design 2011 Edition*. Norwich, UK: The Stationery Office, 2011.

[12] D. Jewell. "Performance Modeling and Engineering". In: edited by Z. Liu and C. Xia. Springer US, 2008. Chapter Performance Engineering and Management Method – A Holistic Approach to Performance Engineering, pages 29–55.

[13] A. Keller and H. Ludwig. "The WSLA Framwork: Specifying and Monitoring Service Level Agreements for Web Services". In: *Journal of Network and Systems Management* 11 (2003), pages 57–81.

[14] H. Liu and P. Crain. "An Analytical Model for Predicting the Performance of SOA-based Enterprise Software Applications". In: *Proceedings of the 30th International Computer Measurement Group Conference (CMG)*. 2004.

[15] Y. Liu, A. Fekete, and I. Gorton. "Design-Level Performance Prediction of Component-Based Applications". In: *IEEE Transactions on Software Engineering* 31 (2005), pages 928–941.

[16] P. Mell and T. Grance. "The NIST Definition of Cloud Computing". In: *National Institute of Standards and Technology – Special Publication* 800–145 (2011), pages 1–3.

[17] N. Milanovic and B. Milic. "Automatic Generation of Service Availability Models". In: *IEEE Transactions on Service Computing* 4.1 (2011), pages 56–69.

[18] A.-C. Orgerie, L. L., and J.-P. Gelas. "Demystifying Energy Consumption in Grids and Clouds". In: *Proceedings of the 2010 International Green Computing Conference*. Chicago, IL, USA, 2010.

[19] C. Patel and A. Shah. *Cost Model for Planning, Development: and Operation of a Data Center*. Technical report. Hewlett-Packard Laboratories Palo Alto, 2005.

[20] E. Pinheiro, W.-D. Weber, and L. A. Barroso. "Failure Trends in a Large Disk Drive Population". In: *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)*. 2007.

[21] *Records of cluster node outages, workload logs and error logs from the Los Alamos National Lab*. Usenix. 2014. URL: `https://www.usenix.org/cfdr-data`.

[22] R. Reussner, S. Becker, E. Burger, J. Happe, M. Hauck, A. Koziolek, H. Koziolek, K. Krogmann, and M. Kuperberg. *The Palladio Component Model*. Technical report. Karlsruhe Institute of Technology, Faculty of Informatics, 2011.

[23] D. Rud. "Performancebewertung und -sicherung von orchestrierten Serviceangeboten". PhD thesis. Otto von Guericke University Magdeburg, 2009.

[24] A. Sachdeva, D. Kumar, and P. Kumar. "Reliability analysis of pulping system using Petri nets". In: *International Journal of Quality & Reliability Management* 25 (2008), pages 860–877.

[25] B. Schroeder, E. Pinheiro, and W.-D. Weber. "DRAM Errors in the Wild: A Large-Scale Field Study". In: *Communications of the ACM* 54 (2011), pages 100–107.

[26] D. Terlit and H. Krcmar. "Generic Performance Prediction for ERP and SOA Applications". In: *Proceedings of the 18th European Conference on Information Systems (ECIS)*. 2011.

[27] K. Trivedi, G. Ciardo, B. Dasarathy, M. Grottke, R. Matias, A. Rindos, and B. Vashaw. "Achieving and Assuring High Availability". In: *5th International Service Availability Symposium (ISAS)*. Edited by T. Nanya, F. Maruyama, A. Pataricza, and M. Malek. Volume 5017. Lecture Notes in Computer Science. Tokyo, Japan: Springer Verlag Berlin Heidelberg, 2008, pages 20–25.

[28] E. Zambon, S. Etalle, and R. Wieringa. "A^{2}thOS: availability analysis and optimisation in SLAs". In: *International Journal of Network Management* 22 (2012), pages 104–130.

[29] V. Zille, C. Bérenguer, A. Grall, and A. Despujols. "Simulation of Maintained Multicomponent Systems for Dependability Assessment". In: *Simulation Methods for Reliability and Availability of Complex Systems*. Edited by P. Faulin, A. Juan, S. Martorell, and J. Ramírez-Márquez. Berlin, Heidelberg: Springer, 2010. Chapter 12, pages 253–272.

Live Migration Overhead Performance Modeling

VMware vMotion Based Study

Mohamed Esam Elsaid, Christoph Meinel

Hasso Plattner Institute
University of Potsdam
{mohamed.elsaid|christoph.meinel}@hpi.de

Cloud computing is the future wave of information technology that provides infrastructure, platform and application as on demand services with low cost and rapid scalability. Infrastructure resources virtualization is the backbone of cloud computing to meet on demand, rapid scalability and resource pooling as the main cloud computing characteristics. Live migration is one of the powerful features in datacenters virtualization. Hosts load balance, power saving, failure recovery and dynamic resource allocation are all dependent on having live migration for the virtual machines. So studying and modeling live migrations is important to predict its impact on the datacenter performance and to take the migration decision at the optimum times. In this paper, we study the impact of live migration on the datacenter resources utilization and power consumption for VMware environment. This overhead modeling can be used to estimate the live migration impact on datacenter resources utilization given the virtual machine and network characteristics. Based on this estimation, the network admin can be alerted with this estimated overhead in order to confirm the live migration request or to postpone it to another optimum time for minimum interruption on the running applications.

1 Introduction

Cloud computing is a promising paradigm for computing and IT services. Now, it is a widely accepted trend for on demand IT resources with elastic scaling and cost efficiency. Cloud computing moves data processing and storage away from desktop and portable PCs into large data centers. To have an on-demand and elastic resources, the IT infrastructure should be virtualized.Virtualization features such as flexible resource provisioning, isolation, cloning and live migration have improved the reliability and utilization of the physical resources. Infrastructure, Platform and Software as a service (IaaS, PaaS and SaaS), are offered using an illusion of availability in resources as demanded by the cloud users.

This virtual availability of resources utilizes the hardware usage, facilitates rapid scaling, and saves power and cost. As a new era for IT services, cloud computing users can get on demand virtual hosts, storage and networking, after specifying their hardware settings, storage capacity, operating systems as well as the installed applications. This is quite different from the earlier infrastructure models where the enterprises had to invest huge cost in building the IT infrastructure resource including cooling and IT staff. Traditional data centers were designed to meet the

peak demand for the running applications; which results in low utilization and waste of resources during the non-peak hours [4].

Live Migration is one of the most important features and a powerful tool in machine virtualization. It allows an entire running and active virtual machine (VM) to be transferred from one physical host to another with a very little interruption which allows seamless movement of online servers in LAN or in MAN scale without asking clients to disconnect and reconnect [9]. Live migration is supported by VMware (vMotion), Xen (XenMotion), Microsoft Hyper-V and Redhat KVM [3]. Servers load balancing, online maintenance, fault tolerance and power saving are all dependent on VMs live migration feature. So live migration is an essential feature in virtual datacenter and cloud computing environment to have dynamic resource management. On the other hand, it is important also to study the drawback of live migration overhead on the datacenter performance.

Live migration cost is classified into energy and performance overhead on the running machines and the live migration execution cost [9]. Execution costs are the total migration time and migration down time. However the physical machines overhead are the increase in CPU utilization, network bandwidth, and power consumption [9]. Migration time and downtime analysis are studied in [13, 3, 9, 7, 2]. Live Migration time and down time are mathematically modeled in [13, 9, 7, 2]. The models in [13, 7, 2] are verified with measurements using Xen test beds. So there is no guarantee if these models are valid for other virtualization tools like VMware; as one of the best performance and commonly used tool in enterprise datacenters [3]. Xen and VMware use iterative pre-copy technique, however the stopping conditions are different; which makes the performance difference [10]. This difference will be more explained in the modeling section of this paper. A very useful comparison between live migration in VMware, Xen, KVM and Hyper-V is done in [3] in terms of the migration time, downtime and migration volume, however there are no models proposed for these results. In [2], RAM intensive, CPU intensive and Disk IO intensive benchmarks are used to measure the migration time with different CPU capacities. However there is no model proposed to estimate live migration impact on CPU and network utilization. To minimize costs, new algorithms are proposed for live migration [10] and [6]. Proposing new live migration algorithms is not the focus of this paper; we mainly study the running algorithms impact on datacenter performance modeling.

In this paper we check the validity of using the proposed migration cost models in [2] for VMware vMotion. The modeled cost in [9] and [7] are the migration time and power consumption. Then we use these models to predict the live migration CPU and network overhead based on regression techniques. Regression techniques based modeling is used in [7] and [2] due to the complexity of live migration modeling. These models can be used to notify the network admin with the live migration impact on the datacenter performance. So the migration request can be confirmed or postponed based on the resources utilization by the running applications in order to avoid service interruption. To the best of our knowledge, no previous work has proposed models to estimate power, network and CPU consumptions for VMware vMotion.

2 Live Migration Time Modeling

2.1 Mathematical Modeling

Live migration in iterative pre-copy technique that is used in Xen and VMware has mainly six phases; as shown in Fig. 1 [10]. These phases are:

1. Initialization: initiating the migration by selecting the VM to be migrated and selecting the target machine.

2. Reservation: the source machine sends a request to the target machine for resources reservation and the target machine answers with an acknowledgment after reserving the required resources for the migration.

3. Iterative pre-copy: the entire RAM is sent in the first iteration, then pages modified during the previous iteration are transferred to the destination. Using shadow page table for memory dirty pages mapping.

4. Stop-and-Copy: When the stop conditions are met, the VM is halted on the source for a final transfer round. The stop conditions in the Xen platform are [13]:

   a. Less than 50 pages are dirtied during the last pre-copy iteration

   b. 29 pre-copy iterations have been carried out9.

   c. More than 3 times the total amount of RAM allocated to the VM has been copied to the destination.

   While the stop conditions for VMware are [8]:

   a. Less than 16 megabytes of modified pages are left.

   b. There is a reduction in changed pages of less than 1 megabyte.

   At the same round of stop- and-copy while transferring the final dirty pages, the migrated VM's CPU state is transferred to the destination.

5. Commitment: the destination host checks if it has received successfully a consistent copy of the migrated VM. Then the target machine sends a message telling the source that it has successfully synchronized the migrated VM states.

6. Activation: after target host informs source host that it has synchronized their states, source VM can be discarded. The migrated VM running on target host is the primary host now and takes over the services offered by source VM.

From Fig. 1, the migration time can be formulated including all its phases details as following:

$$T_{mig} = T_R + T_{pre-copy} + \sum_{i=0}^{n} T_i + T_{checkstop} + T_{stop} + T_{commitment} + T_{trans} + T_{Activation} \quad (1)$$

$$T_R = T_{Prop} + T_{Tran} + T_{Proc} + T_{Prop} + T_{AckTrans} + T_{sync} \tag{2}$$

$$T_{pre-copy} = T_{RO} + T_{copy} + T_{Prop} + T_{Trans} \tag{3}$$

$$T_{Trans} = \frac{V_{Pkst}}{R} \text{ , } T_{Prop} = \frac{L}{S_{Prop}} \text{ , } T_{AckTrans} = \frac{V_{Ack}}{R}$$

$$\sum_{i=0}^{n} T_i = \sum_{i=0}^{n} (T_{prop_i} + T_{Trans_i}) = \frac{V_{mem}}{R} * \frac{1-\lambda^{n+1}}{1-\lambda} \tag{4}$$

$$\lambda = \frac{D}{R} \text{ , } n = \log_\lambda \frac{V_{Th}}{V_{mem}}$$

T_{mig}	Total migration time
T_R	Reservation time
T_{Prop}	Propagation time from source to target
T_{Trans}	Transmission time in the management IP network
$T_{AckTrans}$	Transmission time in the management IP network
T_{proc}	Processing time at target
T_{Sync}	Synchronization time between source and target
T_{RO}	Time for converting source RAM pages to read-only
$T_{commitment}$	Time for checking commitment
T_A	Time taken for activating and handing over the remaining services to the VM on the target and powering it on
V_{Pkt}	Packet volume size in bits
V_{Ack}	Acknowledgement volume size in bits
R	Migration transmission rate in bits/sec
L	Distance length between source and target
S_{Prop}	Signal propagation speed
V_{mem}	VM current memory size during migration
n	Number of migration iterations
D	Dirty pages rate
V_{Th}	Stopping condition threshold volume

From the formulation above, some parameters that impact migration time can be calculated given the environment specifications like T_{Prop} , T_{Trans}, R, V_{Prop}, V_{mem} and D. However, there are also parameters that hardly can be obtained due to complexity such as T_{Prop}, T_{Sync}, $T_{commitment}$ and T_A which might depend on system caching and CPU handling for transactions. So it is worth to use regression techniques as well as the formulation to simplify the relation of the migration time with the main parameters that control it. Migration time formulation (4) is proposed in [7] for Xen environment. As mentioned in stop-and-copy phase, the stopping condition for VMware vMotion is different with XenMotion, so firstly we try to validate if migration time formulation in [7] is valid for vMotion. For models verification and regression techniques implementation, a VMware test bed is used with the following specification in the next sub-section.

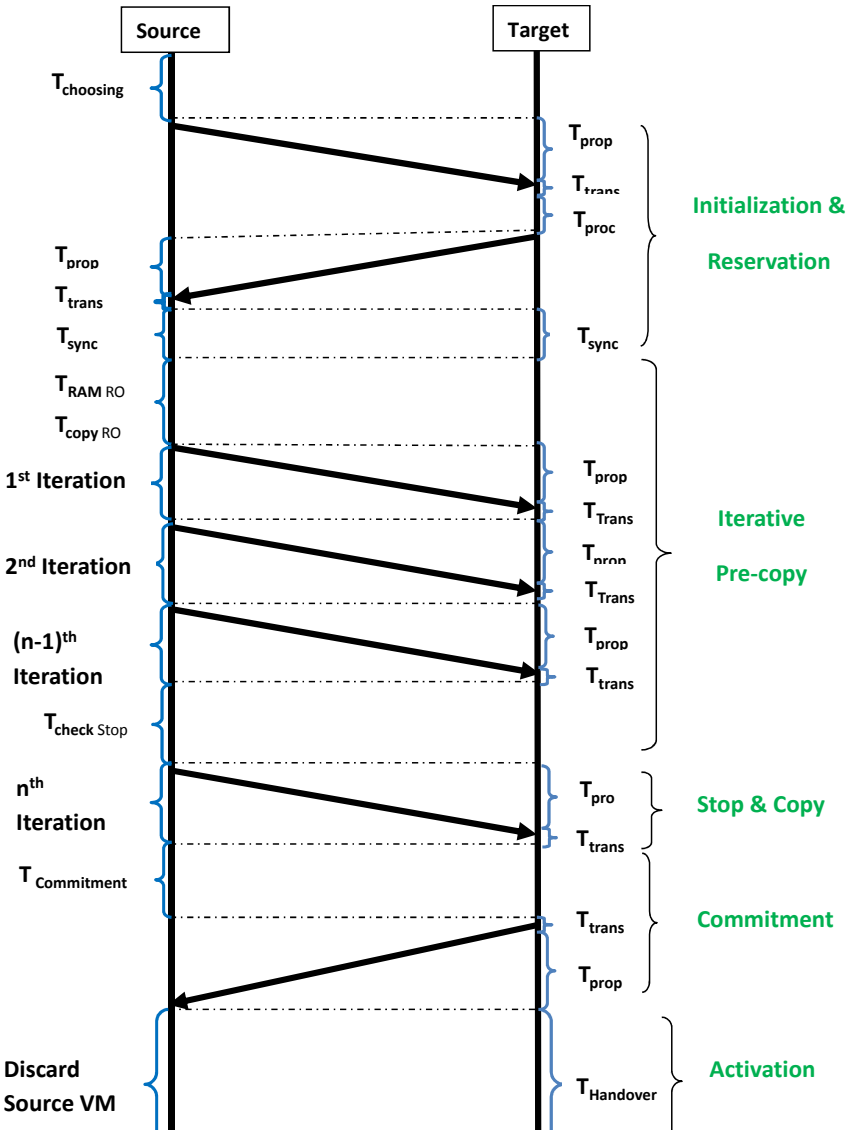

Figure 1: Live Migration Time Diagram

2.2 Testing Environment

The test bed that we have built has 2 Hosts (Dell PowerEdge 2950) with 8 CPU $\times$2.992 GHz Intel(R) Xeon, 4 GB RAM, 4 NICs, 2 HBA with 2 Fiber ports/card and VMware ESXi 5.1 Hypervisor. As shown in Fig. 2; both hosts are connected to shared storage EMC^2 Clariion; 1 TB LUN via FC-SAN. The SAN Switch is Cisco with 4 Gbps ports. The Ethernet switch is Cisco with 1 Gbps ports. Live migration copy iterations are done through the Ethernet switch [12].

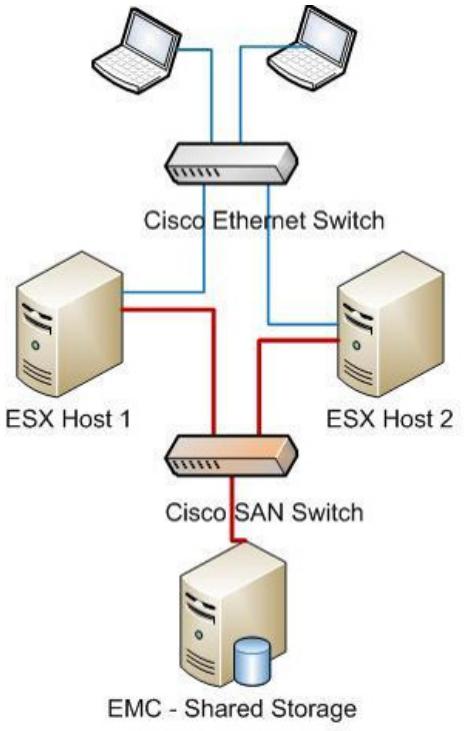

Figure 2: Testing Lab Network Diagram

The two hosts are configured in a cluster that is managed by VMware vCenter Server which allocates the cluster resources and include vMotion feature [5]. Performance parameters values are also gathered using vCenter Server. The VM that is used in migration is Linux Ubuntu 12.04 (32bit) with 4 vCPU. The testing benchmark is Linpack [11] as CPU and RAM intensive benchmark; which is the worst case for a running application. The RAM size is the most effective parameter in the migration performance [9], so we test the impact of live migration on the datacenter performance with different memory sizes to have different migration volumes. Migration volume is the content in total of the CPU and RAM that should be moved from the source to the target host. The VM RAM sizes vary between 1 GB, 2 GB and 4 GB.

2.3 Testing Results

Using the above infrastructure the testing sequence is run as following. The VM is powered on, the benchmark is run and after ten samples at least, the VM migration is started from one of the physical hosts to the other in order to distinguish between the benchmark impact and the migration impact on performance. After the migration is finished, the migration time is calculated and the impact on the target host performance is monitored. Finally the benchmark is stopped. The migration is done 30 times; 10 for 1 GB RAM, 10 for 2 GB RAM and 10 for 4 GB RAM VM. Source host dirty pages are measured to be used in the formulation (4) to calculate the estimated migration time as modeled in [7] but for VMware environment. Calculation results are compared with the average measurements results to know the error of different between modeling and measurements. Comparison results are presented in Table 1.

Table 1: Migration Time Comparison

VM	RAM	Iterative Copy Model (sec)	Measurement Result (sec)	Error %
1	4 GB	104	113	−7.9
2	2 GB	85	105	−19.0
3	1 GB	58	92	−37.0

From the above table, we notice that the results of the proposed model in [7] are always less than the measured results. And the less VM memory size the higher error obtained. The reason for this is the assumption used in [7] in live migration time modeling. This assumption is considering only the iterative copy delay $\sum_{i=0}^{n} T_i$ and ignoring the other phases delay as the iterative copy phase is the most time consuming step in live migration. So the calculated time is always less than measurements. With less memory size the iterative copy phase becomes less dominant and so the error increases.

In order to enhance the mathematical model accuracy, all migration phases delay should be considered. The challenge is how to calculate complex parameters like the processing, synchronization, commitment and activation times. These times are operating system dependent and affected by system caching, so it is hard to be calculated. To overcome this challenge, regression technique is used to have an empirical model for live migration time. Regression techniques were used in [7] and [2] to avoid modeling complexity in power consumption. From the proposed model (1), iterative copy is the most time consuming phase; where the source host memory is transferred to the target host. So we find a relation between the total migration time T_{mig} and $\frac{V_{mem}}{R}$. As shown in Fig. 3, it is obvious that the relation between the migration time and the ratio between migration volume and migration rate. This relation can be simplified in the following formula.

$$T_{mig} = a * (\frac{V_{mem}}{R} + b) \tag{5}$$

Table 2: Migration Time Linear Regression

VM	RAM	Linear Regression Model (sec)	Measurement Result (sec)	Error %
1	4 GB	87.5	113	−22.6
2	2 GB	121.275	105	15.5
3	1 GB	96.0	92	−4.3

Where a and b are constants that depend on the hypervisor management and system caching for migration. For this CPU and RAM intensive testing, $a = 0.9$ and $b = -17$ Linear regression model is compared with migration time measurements to check the model accuracy. This comparison is presented in Table 2; which shows that using linear regression accuracy is acceptable and more accurate than the modeling in [7] but only for 1 GB active memory VMs.

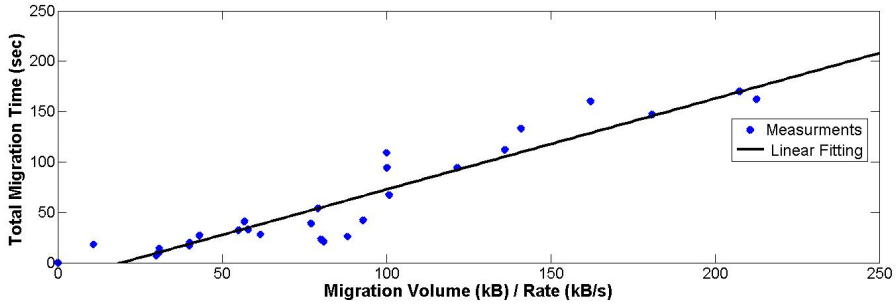

Figure 3: Migration Volume (kB)/Rate (kB/s)

However for larger active memory sizes, the proposed model in [7] is more accurate. So the proposed model for migration time estimation is as following:

$$T_{mig} = a * (\frac{V_{mem}}{R}) + b, \text{ if Active Memory} < V_{Thr}$$
$$T_{mig} = \sum_{i=0}^{n} T_i, \text{ if Active Memory} >= V_{Thr}$$

Based on our testing for CPU and RAM intensive applications, this $V_{Thr} = 2$ GB.

3 Live Migration Power Modeling

Energy consumption modeling is proposed in [9], [7] and [1].In [1], the impact of VMs consolidation on live migration power consumption is discussed. In [9] and [7], it is proven that live migration energy consumption has linear relation with the migration volume. In [9], Xen test bed is used to verify this relation. In this paper, we verify this relation using VMware test bed. In [9] and [7], the relation is between the energy and migration volume. In our testing, the power consumption is measured for each migration; which is the first derivative of energy with respect to time. Migration rate is also the first derivative of the migration volume. Fig. 4 shows an example of power consumption due to one migration request. The proposed model for energy consumption in [9] and [7], is:

$$E_{mig} = c * V_{mig} + d \tag{6}$$

E_{mig}: Migration Energy Consumption in (Joule)
V_{mig}: Total network traffic during migration

Equation (6) is proposed in [9] and [7] for power modeling. In [7], the values of c and d are calculated using linear regression based on testing with Xen test bed;$c = 0.512$ and $d = 20.165$. If the first derivative of (6) is taken the formulation will be as shown in equation (7).

$$P_{mig} = \frac{dE_{mig}}{dt} = c * \frac{dV_{mig}}{dt} = c * R \tag{7}$$

P_{mig}: Migration Power Consumption in (Watt)

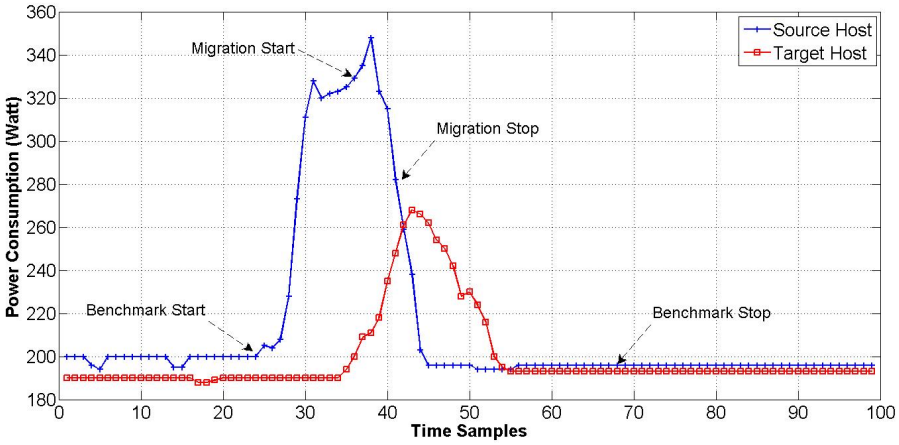

Figure 4: Migration Source and Target Power Consumption

In this paper, we use VMware test bed that is shown in Fig. 2 to verify the linear approximation between migration power consumption and migration rate. Power consumption is measured using vCenter server power and energy charts. After running 30 migrations between the two hosts in the test bed, we monitor the average increase in power consumption due to the migration overhead.

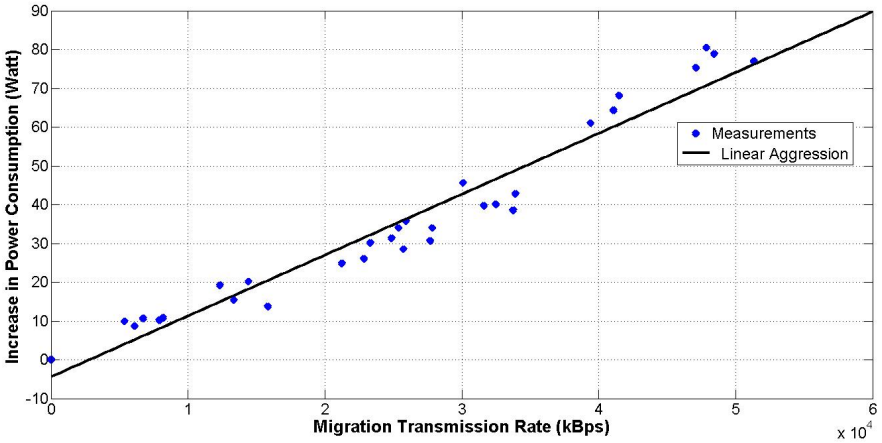

Figure 5: Power Consumption Increase

This power increase is plotted versus the migration rate; as shown in Fig. 5. The relation between the power consumption increase and migration rate can be approximated to a linear relation using linear regression; as shown in Fig. 5. If we apply (7) to the results of Fig. 5, the slope $c = 0.0016$. There is a small constant that can be ignored in the line equation due to its small value ~ 4 Watt compared to the power increase values in the graph.

4 Live Migration Performance Estimation

The models proposed in the formulation section and the model proposed in the related work papers are useful to relate migration cost with other parameters like migration rate or migrated memory volume. In order to make these models more useful, it is important to investigate relations that support the network admins in estimating the migration cost before approving it. So in case that the migration costs are relatively high due to large VMs migration, or datacenter resources are already utilized enough by other applications, it will be recommended to postpone the migration request or to optimize the migration times if it happens automatically. Live migration cost estimation is very supportive to avoid running applications interruption due to the migration overhead. Also, this will reduce the migration process failure rate due to bottlenecks in physical resources. In this paper, we

try to find a relation between parameters that are known for the network admin before migration and estimate the migration cost based on it. For this research approach, we depend on empirical models by testing migration performance for large number of runs and see if this is related to a parameter that we already know before migration.

VM active memory size is one of the key player parameters in live migration cost. This is because the iterative copy of the whole active RAM pages from source to destination. In the next subsection, we show how a phenomenal relation is found between the active memory size before migration and the average increase in the IP network rate from the source to the target host due to migration.

4.1 Network Overhead Estimation

The higher VM active memory size before migration, the more pages should be migrated through the IP network between source and target hosts. To meet the high migration volumes requirements, the IP network algorithm should afford the highest possible transmission rate with keeping low transmission error. Fig. 6 shows one of the migration runs impact on the source host transmission rate and target host receive rate using vCenter Server performance chart; which take a monitoring sample every 20 sec. As shown this increase in the source transmission rate and the target receive rate happens only after starting the VM migration. The rate increase is slightly different between source and target, but the area under each curve is almost the same; which is the migrated volume.

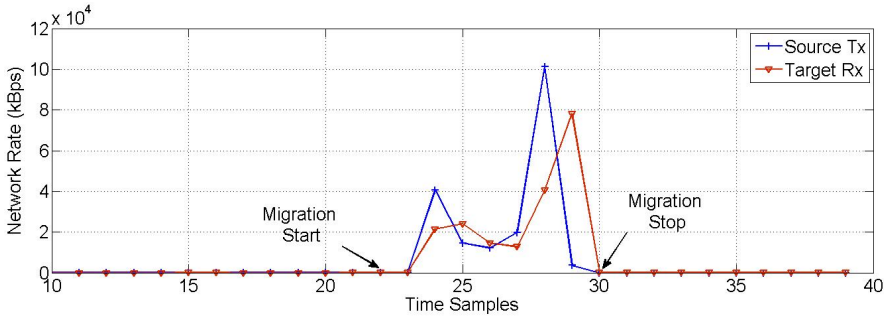

Figure 6: Network Throughput Increase

After running 30 runs, we could notice a fitted relation between the VM active memory size before migration and the transmission rate from the source host; as shown in Fig. 7. This relation is the quadratic approximation of the average transmission throughput increase sample points (8).

$$R_s = \alpha * (V_{mem})^2 + \beta \tag{8}$$

R_s: Source host throughput increase
V_{mem}: Source host active memory size before migration

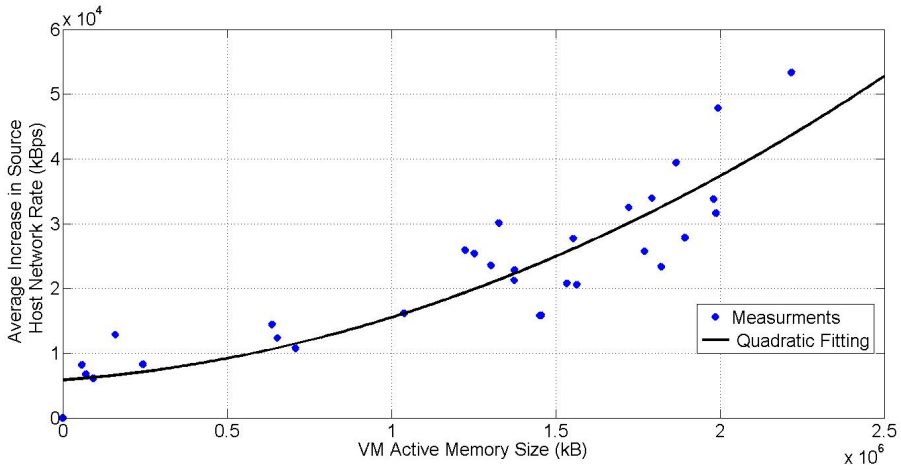

Figure 7: Source Host Network Rate vs VM Active Memory Size

For the curve in Fig. 7, $\alpha = 6 * 10^9$ and $\beta = 6 * 10^3$. The coefficient of is very small, so does not have significance on Rs and can be ignored.

4.2 CPU Overhead Estimation

Live migration consume part of the CPU cycles in the source and target hosts during the live migration process; as shown in Fig. 1. In this part, we study the impact of live migration on the source host CPU utilization and propose a regression based model to predict the CPU overhead in general. CPU regression based model is proposed in [16] to predict the physical host overload in CPU usage and so takes the decision of migrating some VMs to another physical host. In this paper, we use regression techniques to estimate the CPU overhead by the live migration process. After running 10 test for each VM RAM size and so 30 tests in total, the results in Fig. 8 are obtained. The point in Fig. 8 are the peak CPU change after sending VMware vMotion request for each run versus the memory size in kB over the migration rate in kB/s. As shown in Fig. 8, CPU peak overhead point can be approximated to a negative exponential relation 8.

$$C_{peak} = A * e^{-(B\frac{V_{mem}}{R})} + C \tag{9}$$

A, B and C are constants.

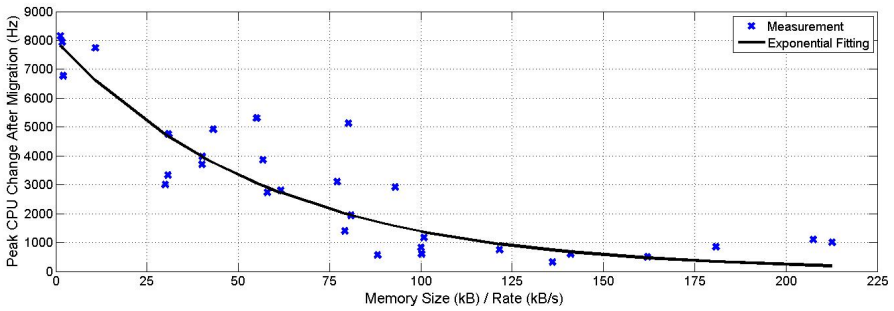

Figure 8: Source Host peak CPU overhead vs VM Active Memory Size Network Rate

For this test: A = 14,000, B = 0.052 and C = 800. From equation (8), the migration rate has a quadratic relation the memory size . If we substitute with according to equation (8), the CPU peak increase will have a negative exponential distribution with the memory size. This means that with increasing the memory size, the CPU overhead increases exponentially.

4.3 Migration Performance Prediction

As shown in Fig. 9, From the results obtained in the section 4.1, equation (8) can be used to estimate the network throughput increase given the VM active memory size V_{mem}. Since the migration average rate R in kB/s is obtained using the VM memory size at the migration time, the migration time T_{mig} can be predicted using (5), the CPU overhead and the power consumption increase can be predicted using (9) and (7).

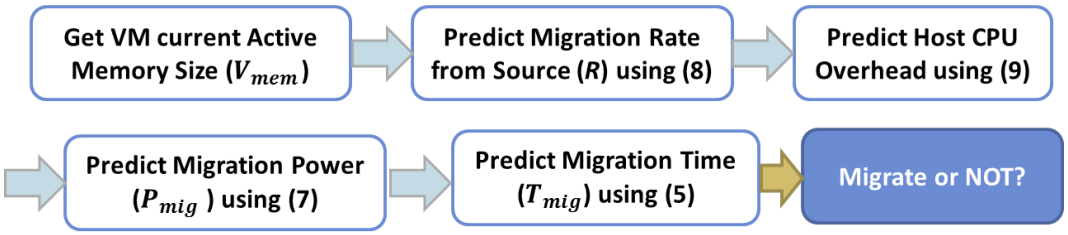

Figure 9: Migration Overhead Prediction Sequence

5 Conclusion

Live migration is one of the essential features in virtualized datacenter environments that drives virtual machines load balance, power saving and fault tolerance.

These resource management techniques are basic for cloud computing in order to have less cost, higher resource utilization, high availability and green IT environment. Live Migration overheads have drawbacks on the datacenter CPU, network, and power consumption.

In this paper, empirical models that are supported by mathematical formulation are proposed to predict the live migration time, power consumption and network throughput before taking the VM migration decision. Based on the proposed prediction model results, the network admin can confirm the migration decision and proceed with it; in case the live migration is a manual action and the prediction results do not cross certain overhead thresholds. In case of automatic load balance or automatic power saving, the prediction model can be integrated with these techniques to optimize the migration decision time. This migration decision optimization should consider the estimated migration overhead and the running applications utilization in order to avoid applications interruption and to minimize bottlenecks probability that may lead to migration process failure.

In the future work, we plan to study the impact of storage migration on the datacenter performance; when the disks content is also migrated.

References

[1] Z. Bose. "The Internet Considered Harmful". In: *Journal of Automated Reasoning* 73 (Dec. 2002), pages 57–68.

[2] E. Clarke, R. Sun, and E. Venkatachari. "The Relationship Between Multicast Solutions and a* Search Using Faluns". In: *Journal of Flexible Epistemologies* 117 (Sept. 1996), pages 48–59.

[3] I. Daubechies. "Comparing Voice-over-IP and Thin Clients with Swarm". In: *Proceedings of VLDB*. Sept. 2001.

[4] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. "Above the clouds: A Berkeley view of cloud computing". In: *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28 (2009), page 13.

[5] F. C. Ito. "Comparing the World Wide Web and XML". In: *Proceedings of SIGMETRICS*. Feb. 1991.

[6] V. Johnson. "Deconstructing Interrupts Using Signet". In: *Proceedings of WMSCI*. Mar. 1997.

[7] G. Kobayashi, H. Sun, and H. Anderson. "An Emulation of Virtual Machines with Feaze". In: *Proceedings of VLDB*. Mar. 2000.

[8] R. Needham, R. Tarjan, and A. Author. "Decoupling Model Checking from Smalltalk in Erasure Coding". In: *Proceedings of the Symposium on Pseudorandom Models*. Nov. 1993.

[9] A. Pnueli. "Comparing the Location-Identity Split and Web Services". In: *Journal of Flexible, Self-Learning Epistemologies* 79 (Jan. 1991), pages 52–68.

[10] D. Ritchie, J. Hennessy, and R. Tarjan. "Deconstructing Massive Multiplayer Online Role-Playing Games with FLIX". In: *Proceedings of INFOCOM*. Jan. 1999.

[11] I. Sutherland. "A Case for Neural Networks". In: *Proceedings of the USENIX Technical Conference*. June 1990.

[12] B. Thompson, N. Wirth, J. Hopcroft, F. Corbato, and U. Takahashi. "Decoupling Kernels from the World Wide Web in the Partition Table". In: *Proceedings of the Symposium on Event-Driven, Replicated Methodologies*. Dec. 2002.

[13] I. Thompson, K. Thompson, D. Zhou, and I. Bose. "Concurrent Symmetries for Object-Oriented Languages". In: *Proceedings of the USENIX Technical Conference*. July 1999.

Quality Attributes for Cloud-based Software Systems

Frank Feinbube, Lena Herscheid, Christian Neuhaus, Daniel Richter,
Bernhard Rabe, Andreas Polze

Operating Systems and Middleware Group
Hasso Plattner Institute
frank.feinbube|lena.herscheid|christian.neuhaus|daniel.richter
bernhard.rabe|andreas.polze@hpi.de

Under the cloud computing term, a vast market of services and products has emerged. The reasons for their popularity are the classic cloud virtues like cost-efficiency, manageability, flexibility, scalability or security. These properties are important for strategic IT planning but difficult to assess and compare in products and existing software systems. In this report, we review methods and tools to assess these quality attributes in cloud-based software systems.

1 Introduction

The concept of cloud computing has ushered in a new era for programming and provisioning of distributed software systems. Traditionally, customized computing infrastructure was installed at the site of customer – from small-scale office groupware solutions to true data centers that serve large companies. On the one hand, this allows the infrastructure to be tailored to the specific requirements of the use case. On the other hand, it burdens the operator with the task of datacenter administration and maintenance. This also presents a large cost-factor, both in up-front investment and running costs. Cloud computing has brought a different concept for provisioning computing infrastructure: instead of buying physical machines, computing resources are offered as services that can be consumed on a pay-per-use billing model. These services are hosted in large-scale data centers, can be accessed over network connection and exist on different levels of abstraction: under the *Infrastructure-as-a-Service* (IaaS) paradigm, compute resources are provided as abstractions physical hardware (i.e. virtual machines). Paradigms like *Platform-as-a-Service* (PaaS) and *Software-as-a-Service* (SaaS) provide services on a higher level of abstraction, where services provide abstract software execution environments (PaaS, e.g. web frameworks) or entire software stacks (SaaS).

This completely changes how software systems are architected and used: the Infrastructure requirements can be treated as a commodity that can be consumed according to current needs and workload. Cloud operators can provide this scalability, as consolidation of computing resources helps to handle the load peaks of individual customers and helps to achieve a better resource utilization. These scale effects also allow cloud operators to offer services at prices which can reduce customer's total expenditure on computing resources. The effort for maintenance

is greatly reduced, as all tasks that concern the software and infrastructure behind the service interfaces are the responsibility of the service provider.

Due to the popularity of the cloud paradigm, a great variety of software products and services has emerged that can be used to provide cloud computing services and create software system based on these services. Different services and software modules can be combined to form a distributed software system. While the amount of products and services provides a great freedom of choice but can also make it hard to see the forest for the trees: Besides mere functionality, the quality of software systems depends on its non-functional properties that describe qualities of operation and development throughout the software lifecycle. The properties are often called the *-ilities and describe attributes such as security, availability, portability, scalability or maintainability (Figure 1).

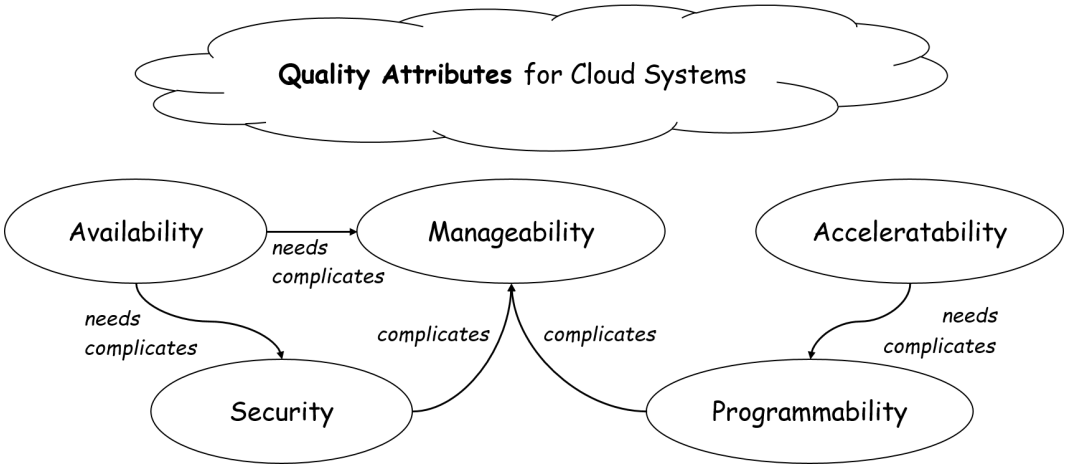

Figure 1: Overview of desired quality attributes for cloud-based software systems

To judge the quality of cloud-based software systems, it is necessary to be able assess these properties in given software systems, products and services. In this report, we give an overview of existing methods that can be used to analyze cloud-based software systems and compare them based on these properties.

2 Availability

A essential requirement for long-running software systems, such as cloud services, is that of *availability*. High availability denotes that a service is very likely up and running at any point in time. This is frequently requested by service level agreements. For instance, a typical high availability requirement – *five nines* (99.999 %) – demands that service downtime per year must not exceed 5.39 minutes. Achieving

high availability is of substantial interest to cloud service providers, since downtime is directly related to a loss of customer confidence and revenue.

However, limited downtime (including both scheduled maintenance and unplanned repair activities) is non-trivial to guarantee. For cloud services, the goal of high availability requires: First, a means for quantifying and comparing availability realistically, and second, the presence of *fault tolerance* (FT) mechanisms to sustain availability despite single node failures.

2.1 Quantifying Availability

Availability is quantifiable as the probability the system delivering its intended service at a given point in time [3].

In systems, where single components can fail at any point in time and repair mechanisms exist, availability is thus a probability distribution in continuous time, resulting from the failure and repair distributions of individual components. $A(t)$ denotes the probability of the system providing its intended service at time t.

Under the assumption that the system is in a stable operational state with fixed failure and repair rates, a static number, the *steady state availability* is often computed:

$$A = uptime/(uptime + downtime) \tag{1}$$

FT techniques can be classified as either *reactive* or *proactive*. Proactive techniques achieve higher availability levels, since downtime is reduced by handling anticipated failures before they occur [29]. However, proactive FT requires the prediction of future failures, relying on an accurate mathematical failure model.

Traditionally, hardware component failure rates have been obtained from field data and stress test results. In a similarly experience-based fashion, failure rates for cloud software should also be based on the dynamic behaviour of the deployed system. For this purpose, monitoring and diagnosis approaches, tailored for the cloud computing paradigm, are needed [12].

2.2 Achieving Availability

In distributed environments such as cloud environments, random infrastructure failures must be anticipated. Therefore, a desired level of availability – that is, tolerance against node failures – can only be achieved by redundancy. The consumers of a cloud service expect *failure transparency*: how a service achieves availability despite single node failures (faults to the overall system) should not matter to the consumer; such failures should go unnoticed.

Hence, a disaster recovery plan needs to 1. detect failures quickly, and 2. provide reliable failover mechanisms while preserving data consistency and confidentiality concerns. This kind of FT needs to be architected carefully for each abstraction layer. Due to its focus on software-based services, the *virtual machine* (VM) and application levels are becoming increasingly important. Here, fault tolerant cloud computing faces novel challenges as opposed to traditional server systems.

Redundancy
Redundancy in traditional server systems has been achieved mainly at the hardware and operating system levels. Many FT mechanisms assume that the nodes are of a fixed number, connected in a fixed topology, and known to each other. Failover to a different machine is more straightforward under these assumptions.

In recent public cloud environments, such assumptions may not hold anymore. In order to achieve scalability, support for dynamic reconfiguration of the underlying infrastructure is necessary. FT needs to become more autonomic and able to handle a dynamically changing infrastructure. While replicating data in a heterogeneous cloud infrastructure, the concerns of availability and security (Section 3) need to be kept in mind simultaneously.

Recovery and Failover
Since cloud services frequently rely on virtualization, the efficient implementation of VM migration and repair techniques is essential. Such techniques can be implemented either at the application side, or at the cloud service provider side. The latter approach is simpler in many cases, since monitoring the VMs can be implemented directly in the hypervisor. In order to avoid data loss during failover, some communication is required to keep backup VMs up to date [34]. Recently, various live VM migration techniques, which avoid VM downtime during failover, have been proposed [28]. They pave the way for promising proactive FT approaches which minimize downtime and maintenance overhead.

At the application level, recovery mechanisms may need to incorporate technologies from different infrastructure providers. This further increases the need for *programmability* (Section 6) and *manageability* (Section 4). Setting up fault tolerant load balancers is a popular approach. Such load balancers (e.g. the Amazon AWS elastic load balancer)[1] automatically scale with the incoming traffic and redirect the requests to working application instances only.

To assess availability, the implementations of replication, recovery and failover – at potentially different levels of the software stack – need to be understood thoroughly. Each underlying mechanism needs to be fault tolerant within itself. Since it has become popular to rely on FT mechanisms on the cloud service provider side, more extensive "availability benchmarks" might be desirable.

3 Security

The security of a software system describes its ability to maintain the confidentiality, availability and integrity of its data and services. These properties a subset of the attributes of dependability [2] and are considered under a fault model that also includes intentional faults, i.e. malicious attacks or misuse. This distinguishes security properties from other attributes of dependability (e.g. reliability, maintainability).

[1] `http://aws.amazon.com/elasticloadbalancing/`, visited on 2014-04-29.

Confidentiality describes a systems ability to protect data from unauthorized access or disclosure. The concept of confidentiality can be abstracted to *exclusivity* [22] an then be applied to data and resources alike. The *availability* property assures that a system functions correctly according to its specification. In a security context, this describes the ability to tolerate intentional attacks on its availability, e.g. robustness against denial-of-service attacks. *Integrity* refers to a systems ability to guarantee the correctness of stored information and prevent unauthorized modification.

Assessing these security properties of computer systems is more important than ever: The distributed character of modern software systems makes many interfaces accessible over network connections and therefore increases the attack surface. In the case of cloud computing large parts are of software systems are no longer under the owners physical control, which enables cloud operators to access the information that is stored on processed on their infrastructure. Methods to analyze the security properties of software systems are therefore required to manage the tradeoff between security risks and the features and benefits of cloud computing. In the following sections, we review qualitative and quantitative methods to analyze the security properties of software systems.

3.1 Qualitative Security Assessment

To describe, plan and compare the security properties of large-scale software systems, several standards have been proposed. These standards provide legal grounds to certify and use computer systems for security-critical applications (e.g. in healthcare, public administration or military applications) and serve as technical guidelines for the development and operation such systems. In the European Union, the *Information Technology Security Evaluation Criteria* (ITSEC) [19] define a set of criteria that independently rate the efficacy of a systems security measures (unsuitable, weak, middle and strong) and the quality of their implementation (E0–E6). While weak measures only provide protection against inadvertent security violations, strong measures should be hard or impossible to circumvent. A similar categorization can be found in the *Trusted System Security Evaluation Criteria* (TCSEC) [7] used in the US. In the strictest security levels (E6 in ITSEC, A in TCSEC), a formal verification of the system is required (see below).

In practice, the security properties of computer system is often influenced by tradeoffs between security on the one hand and functionality, usability and efficiency on the other hand [6]. Additionally, as the code-base of software systems increases, bugs cannot always be avoided [14] an inevitably introduce vulnerabilities [13]. Consequently, threats to security exist in every computer system. It is therefore important to identify and understand these threats. Microsoft *STRIDE* [30] represents a structured approach to identify the threats present in a software system. This tool-aided analysis is based on a data-flow diagram representation of the software system. Based on this model, the tool automatically identifies security threats that apply to processes, data stores and network links of the system. The result of the analysis is a list of threats which can be rated and annotated with present threat mitigations.

The most rigid form of security analysis is *provable security*. In theory, a security property can be linked to a set of states of a software system in which the security properties is fulfilled. If the behavior of the system can be predicted for all possible use cases and circumstances, all reachable states can be predicted. If "insecure" states are not reachable, the software system is proven secure. Examples for provable security security proofs for cryptographic protocols (see e.g. [21, 26]) or information flow theory (see e.g. [33]). However, provable security is usually limited to small programs or single algorithms and cannot capture the complexity of large-scale software systems.

3.2 Quantitative Analysis Methods

In reality, security remains a tradeoff between efforts to protect resources and circumvent security measures. Qualitative methods alone are not sufficient to capture the probabilistic nature of security. To describe individual security properties, quantitative measures are indicate a probability that a given security property is fulfilled in a given time interval under given circumstances. This allows to include security properties as quantities of the software engineering process: security measures become part of the software requirements specification and are tested as part of the requirements verification.

Quantitative security assessment is usually an analysis in the failure space: individual threats present in the system are treated as *security faults* that threaten a specific security property [24]. Using mathematical models, simulations and field data, probabilities can be derived for individual threats to be successfully used for attacks. By combining the probabilities obtained for individual threats, an overall probabilistic measure for individual security properties can be calculated (see e.g. [23]).

Different methods and models have been proposed for quantitative analysis of security properties. A commonly used method are *attack trees* [37], which are derived from *Fault Tree Analysis* [10]: Threats are denoted as leaves in a tree, while the analyzed security property is at the root. Attack trees model the logical combinations of threats that could harm the security properties as a network of logical connections (AND/OR) that connects the root to the leaves. If probabilities for the leaves are available, the overall probability for the security property under analysis the be violated can be calculated. If an order of events determines the success of security attacks, state-based models can be used (see e.g. [8, 4]). Where state-explosion prevents state-based modeling, simulations can be used to analyzed security-relevant processes or protocols (see, e.g. [35, 36, 38]).

4 Manageability

Operating IT-infrastructures is a challenge that has to be solved in companies and cloud providers in particular. To master the challenge a powerful set of tools is required, that may be summarized in manageability. The *Information Technology*

Infrastructure Library (ITIL)[18] list some well known rules and best practices in general, but does not covers fast changing cloud computing.

In todays cloud computing systems we have two views for assess manageability: the provider and the customer view. Both views have different needs in management.

4.1 Provider View

The provider of cloud computing resources and services needs management tools to operate the infrastructure and offer solid services to customers. Section 2 and 3 outlined properties that have to met. These management tools need to fulfill some basic tasks: monitoring, deployment, and configuration.

Monitoring is required on two levels: the infrastructure and the services. The infrastructure that runs the services has to be monitored to detect failures, overload situations or unused resources. Services rely on the infrastructure but monitoring is required to check service health, billing and customer offering. Deployment is an integral part for manageability of cloud computing, but they're different kinds of deployment: deployment of infrastructure services (e.g. additional server) and deployment of cloud computing services (e.g. customer service). Configuration is modifying a service without new deployment (e.g. start, stop).

While most of managing infrastructure services implies human interaction, is management of cloud computing services automated. That's the only way how to operate cloud computing offerings in large and also implies a stable manageability.

4.2 Customer View

A customer has interfaces to use the cloud computing services. These are service dependent and vary among providers. Limitations in manageability have to be accepted and can't changed in general. The cloud computing models offer different possibilities in management. The SaaS model has only a few settings that can be managed because the entire service is self-contained. At PaaS level the customer has the capabilities to modify his services (e.g. start, stop, deploy, publish). In IaaS the variety of management support is biggest. Amazon offers with CloudWatch an separate service to monitor and manage resources in the EC2[1]. HP Cloud[16] relies on OpenStack[27] and supports the standardized interfaces. It is also possible to deploy own machine images with built-in management solution, but limited to operating system level support.

Management interfaces for the customer have to meet the self-service model, where the services can easily monitored, deployed and configured to fulfill its demands.

But the new world is hybrid! That means management of IT-infrastructures has to cover not only single cloud computing model and a single public cloud. Today companies can use hybrid cloud solutions[15] to combine on-premise infrastructure and public cloud offerings. In that sense the future management has to support hybrid services. Services may cover different cloud computing offerings that are defined in orchestration languages (e.g. Hewlett-Packards Cloud Maps[15]).

But if this is extended to all cloud computing models and public cloud offerings a new model for management and open interfaces is required to support manageability of future IT-infrastructures.

5 Acceleratability

Clouds are considered to deliver 'pluggable' computing power. Thus they are used for performance-hungry applications that involve a lot of number crunching on state-of-the-art hardware. Due to physical limitations [11] modern CPUs are accompanied by various types of accelerators [5, 25, 17] to account for the ever increasing computational demands.

Besides the GPU compute chip that is integrated into modern Intel CPUs [20], dedicated accelerator cards like NVIDIAs Tesla product line [25] and Intels Xeon Phi card [17] are of particular interest for the industry.

When it comes to virtualization of accelerators, the following characteristics are to be considered [9]:

- *Performance:* indicates the execution speed. Depends primarily on the overhead introduced by the virtualization mechanism.
- *Fidelity:* a measure of the number of features of the physical device that are supported by the virtualized device.
- *Multiplexing:* indicates the degree of multiple VMs sharing the same physical GPU.
- *Interposition:* degree to which access of the a virtual machine to the physical hardware can be mediated. Interposition enables features like live migration, hibernation, and fault-tolerant execution.

Keeping these characteristics in mind, let's take a look at the cloud offerings that are currently provided by cloud vendors:[2]

- *Amazon Web Services:* hosted GPUs and GPU Cloud. Only in North America.
- *Nimbix Informatics Xcelerated:* hosted GPUs and GPU on Demand. Only in North America.
- *Peer 1 Hosting:* hosted GPUs and GPU Cloud. North America and Europe.
- *Penguin Computing:* hosted GPUs. Only in North America.
- *RapidSwitch:* high Performance Computing and GPU Cloud. Only in Europe.

[2] `http://www.nvidia.com/object/gpu-cloud-computing-services.html`, visited on 2014-04-29.

- *Softlayer:* high Performance Computing, Hosted GPUs and GPUs on Demand. North America, Europe and Asia.

Available offerings grant virtual machines direct access to the GPU card that is built into the server the virtual machine is running on. This allows a single guest operating system to enjoy a maximum of performance and fidelity, while inhibiting multiplexing over multiple virtual machines and disallowing any cloud features relying on interposition.

The reason for this situation is the fact that with the exception of Microsoft Hyper-V, which has no GPU virtualization support whatsoever, currently only pass-through GPU virtualization is supported by all the major vendors of virtualization technologies. [31]

Since Virtual Desktop Infrastructure (VDI) is a strong focus for current GPU technology developments, emerging GPUs introduce the concept of vGPU (virtual GPU) [39] which allows to partition a GPU and use it by multiple virtual machines. A physical GPU can be divided into at most 32 virtual GPUs of the same size. Citrix XenServer already supports this technology.

While static fixed-size partitioning helps, real multiplexing can only be achieved when timesharing and oversubscription are enabled as well. Only then will it be possible to save resources by consolidating applications. Interposition features like live migration are also needed for a seamless integration of accelerators into the cloud, but are still unavailable.

In order to be competitive clouds must target two conflicting objectives: firstly, they need to deliver the best experiences to their customers, while secondly consolidating virtual machines so that they can make the best use of their resources. The first objective is clearly related to performance and fidelity. It can be achieved with the current virtualization solutions of allowing guest operating systems direct access to the underlying accelerator hardware.

For the second objective on the other hand we need a better support for multiplexing and interposition. Key players in the industry have already identified this challenge and are actively working towards applicable solutions, the partitioning provided by NVIDIAs virtual GPU being the first step towards fully cloud-capable accelerator technologies.

6 Programmability

While SaaS is a cloud computing model that already offers ready-to-use applications to end-users, IaaS provides on-demand hardware and operating-system services as well as PaaS delivers application platforms an solution stacks. So with cloud-based software systems developers do not have to care about issues such as scalability and availability while the application's workload is increasing. With resources provisioned on-demand, operating IT systems is made more cost effective and technically flexible.

Programmability of a cloud system means for us the degree of flexibility to define, extend, and reconfigure a cloud application's behavior and structure as well as the

ability to integrate a cloud system into the software lifecycle ecosystem. Within cloud bases systems, some programmability attributes can roughly be categorizes as follow: [40] [32]

- The *logic layer* is where the business logic is executed.
- The *data layer* manages data processed by the logic layer. It is responsible for storing and loading required data as well as caring about data consistency within a distributed cloud environment.
- The *infrastructure layer* is where both the logic layer and the data layer are operated, monitored, and managed.
- *Crosscutting concerns* will cover some most or all layers mentioned above.

6.1 Infrastructure Layer

IaaS cloud providers usually offer different kinds of operating systems with preconfigured settings. Virtualization technologies make it possible to create a repository filled with VM-images for a wide range of purposes. It is even possible to create whole environments with custom settings for network, compute, and storage resources – based upon templates (see HP CloudSystem[15]). On the contrary, within PaaS environments the chance to influence the structure of the underlying infrastructure layer is limited or impossible – in exchange a developer or administrator does not have to manage and maintain these components.

Attributes: provided hardware resources (CPU, memory, storage, network), resource performance and scalability, provided operating systems, supported orchestration operations, customization (load balancing, auto-scaling, user defined performance)

6.2 Logic Layer

The logic layer consists of two parts: the application language and the application framework. The important attribute for this layer is the *support for technology platforms and stacks*.

Either a developer writes native applications targeted to a specific operating system, or he uses specific runtime environments such as Microsoft .NET or Java. Depending on the infrastructure layer it is may possible to deploy stand-alone applications to the cloud environment, or a developer is bound to an application server such as Apache Tomcat or Microsoft IIS. To achieve a high programmability it is also important to know whether standard libraries are available (or installable), whether third party components are available (or installable), or whether direct resource access (e.g. local file system, threads, outbound network connections) is possible.

Attributes: programming language, supported frameworks (e.g. .NET, Tomcat, JavaSE, JavaEE, standard libraries), deployment packages (e.g. WAR, EAR, DLL), direct resource access

6.3 Data Layer

The data layer is responsible for the storage and retrieval of data. It manages the access to the data from potentially multiple different components and cares about data synchronization and consistency.

Nowadays, there are two popular types for data storage: Relational databases with a data schema (e.g. SQL databases) and document databases with more unstructured data (e.g. NoSQL databases, key-value databases, big data stores, message queues, distributed hash tables). The choice for one of them heavily affects the tools, methods, and algorithms a developer can use. To change this decision later mostly needs a huge change in an applications structure and behavior. Also these two options differ in the ability to deal with evolving data schemata.

Attributes: number of data items, read/write velocity, data quality (structured, unstructured), data schema, database scalability (clustered, master/slave, single instance, content delivery network), consistency constraints, migration

6.4 Crosscutting Concerns

Finally, there are some attributes that effect the programmability of a cloud system and touch most or all layers mentioned above: The availability of development tool (command line tools, source control integration, web-based console), the availability of testing environments (development, testing, stating, production), access to log files, the possibility to debug running applications, integration into build and deployment systems, and documentation.

7 Conclusion

As an emerging computing paradigm, with novel and large scale technical challenges to face, quality attributes for cloud systems need to be well understood in order to evaluate different systems, and to discuss the trade-offs and degrees of freedom for designers of cloud applications.

We have proposed five distinct non-functional quality attributes for cloud systems, namely:

- **Availability** of the service, even in the presence of node failures and byzantine faults;
- **Security**, the robustness of the system against attacks at different levels in the software stack;
- **Manageability** of resource and job allocation, despite increasingly heterogeneous applications and infrastructure;
- **Acceleratability** by providing access to accelerator hardware such as GPUs to customers in an efficient way;

- **Programmability**, in order for the system to be extensible and re-configurable during runtime.

The different quality attributes relate to each other in different ways. They may be in trade-off relationships, such as the trade-off between easy manageability and high security, or they may be pre-conditions for one another, for instance, security is a precondition for availability in attack scenarios. It is important to understand the relations and interactions between these attributes thoroughly in order to ensure high quality cloud applications. Further, the attributes we have named all play a role at various levels in the software stack. Engineers should be aware of this and design the software with each of the above named attributes in mind.

We further believe that efforts towards quantifying and further formalizing cloud system quality attributes is necessary. Quantification is essential to characterize different cloud systems in a comparable fashion, and to provide a standardized measure of quality for them.

Taking this discussion (summarized also in Figure 1) as a starting point, research from different software engineering communities should overlap more, in order to provide a unifying framework for assessing the quality of cloud applications.

References

[1] Amazon. *Amazon Elastic Compute Cloud (Amazon EC2)*. Aug. 2014.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. "Basic concepts and taxonomy of dependable and secure computing". In: *Dependable and Secure Computing, IEEE Transactions on* 1.1 (2004), pages 11–33.

[3] A. Avizienis, J.-C. Laprie, B. Randell, et al. *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science, 2001.

[4] F. Besson, T. Jensen, D. L. Métayer, and T. Thorn. "Model checking security properties of control flow graphs". In: *Journal of computer security* 9.3 (2001), pages 217–250.

[5] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli. "State-of-the-art in Heterogeneous Computing". In: *Sci. Program.* 18.1 (Jan. 2010), pages 1–33.

[6] L. F. Cranor and S. Garfinkel. "Guest Editors' Introduction: Secure or Usable?" In: *Security & Privacy, IEEE* 2.5 (2004), pages 16–18.

[7] Department of Defense. *Trusted Computer System Evaluation Criteria*. Dec. 1985.

[8] D. Dolev and A. Yao. "On the security of public key protocols". In: *Information Theory, IEEE Transactions on* 29.2 (1983), pages 198–208.

[9] M. Dowty and J. Sugerman. "GPU virtualization on VMware's hosted I/O architecture". In: *SIGOPS Oper. Syst. Rev.* 43.3 (July 2009), pages 73–82.

[10] A. Ericson and C. Ll. "Fault tree analysis". In: *System Safety Conference, Orlando, Florida*. 1999.

[11] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. "Dark Silicon and the End of Multicore Scaling". In: *SIGARCH Comput. Archit. News* 39.3 (June 2011), pages 365–376.

[12] I. Foster, Y. Zhao, I. Raicu, and S. Lu. "Cloud computing and grid computing 360-degree compared". In: *Grid Computing Environments Workshop, 2008. GCE'08*. Ieee. 2008, pages 1–10.

[13] S. Frei, M. May, U. Fiedler, and B. Plattner. "Large-scale vulnerability analysis". In: *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*. ACM. 2006, pages 131–138.

[14] J. E. Gaffney. "Estimating the number of faults in code". In: *Software Engineering, IEEE Transactions on* 4 (1984), pages 459–464.

[15] Hewlett-Packard. *CloudSystem*. 2014.

[16] Hewlett-Packard. *HP Helion Public Cloud*. 2014.

[17] Intel. *Intel® Xeon Phi™ Coprocessor Developer's Quick Start Guide*. 2013.

[18] IT Infrastructure Library. *ITIL® Home*. Aug. 2014.

[19] ITSEC. *Information Technology Security Evaluation Criteria (ITSEC): Preliminary Harmonised Criteria*. Technical report. Commission of the European Communities, 1991.

[20] M. Kelly and K. Hartman. *Intel Ivy Bridge Architecture*. 2013.

[21] N. Koblitz and A. J. Menezes. "Another look at" provable security"". In: *Journal of Cryptology* 20.1 (2007), pages 3–37.

[22] C. Meadows. "An outline of a taxonomy of computer security research and development". In: *New Security Paradigms Workshop: Proceedings on the 1992–1993 workshop on New security paradigms*. Volume 1993. 1993, pages 33–35.

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

[24] D. Nicol, W. Sanders, and K. Trivedi. "Model-based evaluation: From dependability to security". In: *Dependable and Secure Computing, IEEE Transactions on* 1.1 (2004), pages 48–65.

[25] NVIDIA Corporation. *NVIDIA's Next Generation CUDA Compute Architecture: Fermi*. 2009.

[26] K. Nyberg and L. R. Knudsen. "Provable security against a differential attack". In: *Journal of Cryptology* 8.1 (1995), pages 27–37.

[27] OpenStack Foundation. *The OpenStack Project*. 2014.

[28] P. D. Patel, M. Karamta, M. D. Bhavsar, and M. B. Potdar. "Article: Live Virtual Machine Migration Techniques in Cloud Computing: A Survey". In: *International Journal of Computer Applications* 86.16 (Jan. 2014). Full text available, pages 18–21.

[29] A. Polze, P. Troger, and F. Salfner. "Timely virtual machine migration for proactive fault tolerance". In: *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*. IEEE. 2011, pages 234–243.

[30] B. Potter. "Microsoft SDL Threat Modelling Tool". In: *Network Security* 2009.1 (2009), pages 15–18.

[31] S. Rahim. *State of GPU Virtualization for CUDA Applications 2014*. July 23, 2014.

[32] R. Ranjan and B. Benatallah. "Programming cloud resource orchestration framework: operations and research challenges". In: (2012). arXiv: 1204.2204 [cs.DC].

[33] A. Sabelfeld and A. Myers. "Language-based information-flow security". In: *Selected Areas in Communications, IEEE Journal on* 21.1 (2003), pages 5–19.

[34] D. J. Scales, M. Nelson, and G. Venkitachalam. "The design of a practical system for fault-tolerant virtual machines". In: *ACM SIGOPS Operating Systems Review* 44.4 (2010), pages 30–39.

[35] A. Schlosser, M. Voss, and L. Bruckner. "Comparing and evaluating metrics for reputation systems by simulation". In: *A Workshop on Reputation in Agent Societies as part of*. Citeseer. 2004.

[36] A. Schlosser, M. Voss, and L. Brückner. "On the simulation of global reputation systems". In: *Journal of Artificial Societies and Social Simulation* 9.1 (2006).

[37] B. Schneier. "Attack trees". In: *Dr. Dobb's journal* 24.12 (1999), pages 21–29.

[38] V. Venkataraghavan, S. Nair, and P. Seidel. "Simulation-based validation of security protocols". In: *Proceedings of OPNETWORKS 2002* (2002).

[39] W. Wade and I. Williams. "NVIDIA Grid: State of the Art in Virtualized Graphics". In: *SIGGRAPH 2013*. 2013.

[40] C. Weinhardt, A. Anandasivam, B. Blau, N. Borissov, T. Meinl, W. Michalk, and J. Stoesser. "Cloud computing–a classification, business models, and research directions". In: *Business & Information Systems Engineering* 1.5 (2009), pages 391–399.

Self-Configuring Data Imports for SAP HANA Cloud Environments

Hendrik Müller, Matthias Splieth, Sascha Bosse and Klaus Turowski

Very Large Business Applications Lab
Otto von Guericke University Magdeburg, PO Box 4120, Magdeburg
{hendrik.mueller|splieth|sascha.bosse|klaus.turowski}@ovgu.de

In cloud environments, customer-triggered data imports need to be configured dependent on information, which is usually not accessible for customers. In this paper, this problem is addressed by developing a concept for automatically configuring data imports in order to optimize their import speed. The concept relies on a continuously growing knowledge base that can be used for determining suitable parameters for new imports. In order to provide data for the knowledge base, we automated the execution of 878 imports with a total amount of 34 TB of data for testing all possible parameter permutations within a given range, and stored the corresponding results. Subsequently, we were able to perform a self-configuring import of an additional dataset by querying the knowledge base for the most suitable combination of import parameters based on the similarity of the additional dataset to the datasets in the knowledge base. For the purpose of the evaluation, we calculated the number of inserted rows per second and ascertained an improvement of 84.89 % for the self-configuring import compared to the default configuration, and an improvement of 15.47 % compared to an import configuration recommended by SAP.

1 Introduction

In modern times, companies need to be able to rapidly respond to the needs of and changes in the market in order to remain competitive. In this context, the analysis of data has proven to be important for decision support in various domains. However, due to the proliferation of a growing number of data sources, the amount of data that needs to be analyzed in order to support decision making is growing rapidly [7]. But in order to be able to derive knowledge from the huge amount of data, scalable analytic services are needed which require high-performance systems [11], such as provided by cloud environments. Therefore, it is suitable to analyze big data using web-based cloud services enabling data mining algorithms to be applied on multiple datasets and leveraging economies of scale. Hence customers can benefit from fast results without having to maintain potentially expensive hardware. Offering a multi-tenant database, analytical capabilities and a built-in web server, SAP HANA is used to develop the concept that is presented in this paper. In the course of this contribution, we focus the first stage of such big data cloud services which implies the import of datasets from different sources, potentially at the same time. Assuming the customer's datasets to be available as files containing comma-separated values (CSV), we leveraged the bulk load feature of SAP HANA in order to import these files. Imports can be parameterized depending

on available hardware resources, the current system load, dataset properties and customer-specific service level agreements (SLA). Database administrators (DBA) are able to access this information, but inside cloud environments, data imports are triggered by customers. In order to be able to quickly import data, the parameterization of imports needs to be automated since customers cannot access the same information as DBAs. In case of high utilization, it might be suitable to adjust the number of parallel imports in order to maintain a stable system state and not affect the overall performance in terms of running analytics and other import jobs. Therefore a separate staging area is required that serves as a buffer in order to detach user-interactive file uploads from load-dependent data imports.

In this contribution, we present a concept for cloud-based data imports by considering this aforementioned requirement as a first step toward big data analytics as a service. To enable data imports to be self-configuring and fully automated, the proposed concept involves a performance knowledge base for data imports of different sizes for all database instances of a HANA cloud environment. All import jobs are controlled and logged by means of stored procedures and tables inside a relational database schema. By storing the resulting number of inserts per second as a key performance indicator (KPI) for each import job and by associating it with the respective import parameters, (system, hardware and dataset information), we built an architecture that allows querying for best performing configurations when invoking similar imports. The architecture scales with the cloud infrastructure and changing data properties since every import contributes to the shared knowledge base, which can be made available to all database systems by means of SAP HANA's smart data access feature.

The proposed concept was developed and implemented in terms of a proof-of-concept following a design science approach [4]. The concept is described in section 2 by means of an exemplary scenario. Subsequently, the technical architecture of the SAP HANA cloud environment that is used for this purpose is outlined. In order to be able to evaluate the designed concept, initial KPI values for different database servers, datasets and import configurations are required. Thus, we conducted 878 imports within three experiments by automating the imports for various parameter permutations as presented in section 3 in order to build a knowledge base that comprises of these KPI values. Subsequently, the proposed concept is evaluated in section 4 by using the knowledge base for determining the import parameters that are to be used for an additional dataset. In section 5, the paper is summarized and, furthermore, an outlook on future research is given.

2 Data Imports for SAP HANA Cloud Services

When integrating big data and cloud services, the three dimensions variety, velocity and volume [6] will need to be faced by respective providers. Term velocity refers to both the time that is needed to process data and the time in which new data is generated. Thus, a growing amount of data needs to be imported from different sources within an acceptable period of time by a cloud service provider.

2.1 Scenario Overview

The import of CSV files can be initiated from both the server and the client by using SAP HANA Studio. When importing data onto the server, cloud service providers are able to trigger and parameterize the import of the data autonomously, depending on system utilization and SLAs, which may be derived from pricing strategies. Therefore, we split the import process into two sub-processes: "uploading CSV-formatted data files to a staging area represented by one or more gateway servers (step 1)" and "importing data files into the appropriate database system through self-configuring imports (step 2)". The first sub-process involves user interaction in terms of uploading the data files that are to be imported, whereas the second sub-process is triggered and configured automatically by determining the performance-influencing parameters "threads" (number of parallel threads for the concurrent import) and "batch" (number of rows imported for each committed unit).

The experiments performed in the course of this contribution (confer section 3 and section 4) indicate that there is no generic parameter combination of "threads" and "batch" (hence referred to as "import configuration") that archives the best result in any case. In fact, suitable import parameters seem to depend on the characteristics of the dataset such as the file size, the number of rows and the number of columns. Furthermore, suitable import parameters depend on hardware characteristics of the respective host, such as the CPU utilization. For this reason, we parameterize import jobs based on historical data. In order to be able to use historical data, the relevant information concerning an import needs to be stored. This includes, for example, the parameters that were used for the import (such as batch and threads), the properties of the dataset (such as its size) and information on the database host . When new imports are to be processed, the stored information is queried for previous results of similar imports. The similarity of previous imports is thereby determined according to a string metric that relies on the properties of previously imported datasets. The corresponding parameters of a previously conducted import that led to a high performance regarding the import speed can then be used to parameterize the new import. Accordingly, the same information is stored for new imports after they will have been finished. Hence, the knowledge base is extended with every new import and scales with a growing amount of data as well as changing dataset properties, since the probability of finding results for similar, previously conducted imports increases with every new import. For implementing the knowledge base, we created a relational database schema and corresponding features that can be used in order to automatically import the data uploaded by customers.

2.2 Technical Architecture

The system environment that was used for evaluating the designed concept comprises three servers, each of which running a single SAP HANA instance. In order to increase the cloud provider's flexibility, the storage is abstracted from the servers.

Thus, all the data is stored on a central network attached storage (NAS) so that HANA instances can be started on any server following HANA's tailored data-center approach (cf. [10]). Relocations can be performed by mounting the log and data volumes to different servers. In addition to these servers, we configured an additional node to operate as a gateway that provides access to the previously described staging area. Data files that are uploaded by customers will be stored on the NAS through the gateway server and will therefore be accessible by all HANA instances for a server-triggered import, using a redundant 10 GB Ethernet connection. Figure 1 illustrates the components of the architecture.

For logging all import performance results as well as respective import properties and parameters, we created a database schema, which stores all information that forms the knowledge base. Besides several procedures that are used to process the stored information, it comprises of the tables illustrated in Figure 2.

The table "Result", which is positioned in the center of Figure 2, contains the start time and end time for each performed import. The resulting values "duration" and "instertspersecond" are calculated and updated as soon as an import finishes, whereas "insertspersecond" refers to the average number of inserted rows of the corresponding dataset per second. By joining the result with the associated tables "Dataset", "Host", "Threads" and "Batch", the corresponding parameters as well as the properties of the imported dataset and the server that processed the import can be determined. In this manner, the parameters that led to the best imports can be identified when configuring the import of a new dataset.

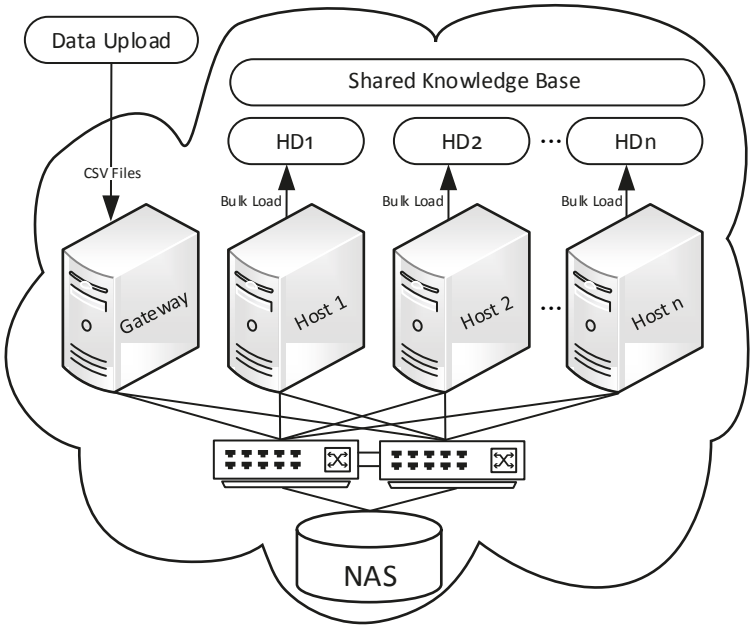

Figure 1: Technical Infrastructure of the Concept

However, initial data is needed for the knowledge base in order to be able to query for suitable import parameters. For generating initial values, we inserted tuples for "threads" and "batch" within a defined range and performed experiments by running imports parameterized with any possible permutation of threads and batch, which are computed by a stored procedure.

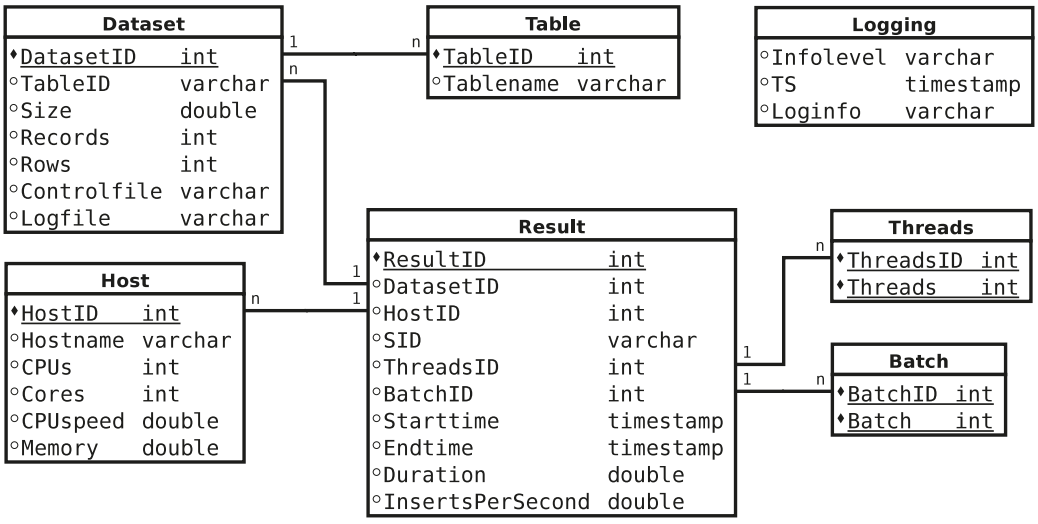

Figure 2: Database Schema that Forms the Knowledge Base

3 Import Experiments

In order to verify the architecture proposed in the previous section and in order to provide initial data for the knowledge base, several experiments were conducted for each of the imported datasets. In each experiment, multiple iterations are performed over the permutations until the best result in terms of inserts per second is found. Provided that outliers occurred in the course of an iteration, it was manually repeated for the particular outlier. Afterwards, the mean values were computed and inserted into the "result" table. This procedure leads to a knowledge base that can be queried for suitable parameters when new datasets are to be imported. The experimental results and the experimental setup are presented in the following.

3.1 Experiment Design

In the case of a new import, the historical data that is stored in the knowledge base can be queried for a similar dataset. The associated parameters, a combination of *(Threads, Batch)*, of the best result in terms of the highest number of achieved inserts

per second can then be used to parameterize the new import. In order to get to the required historical data, numerous experiments are conducted. The corresponding information is stored as described in subsection 3.2. Since a manual execution of the experiments would be very expensive regarding the vast amount of permutations of the import parameters, the import process has been automated. This allows the automatic testing of different combinations of parameters and, subsequently, the automatic determination of the import results. Figure 3 shows the components that were implemented for this purpose.

The experiments were configured and controlled by means of database objects in terms of different tables, views, and a stored procedure. The stored procedure `prepare_load` uses different input parameters to automatically generate the import statements and the corresponding statements for determining the results of the import, such as the duration and the number of inserts per second achieved with the respective parameters.

The procedure generates import statements rather than immediately importing the data since the auto-commit feature that is used by `IMPORT FROM` statements is not allowed in `BEGIN ... END` blocks. The parameter Dataset provides information about the dataset that is to be imported, such as the number of rows and the name of the target table. `Batch` and `Threads` are the parameters that are varied in the course of each iteration in order to investigate their effects on the load performance and to store the corresponding results in the database. `M_Views` are public system views, which provide different information about the invoking system, such as the HANA system identifier and the hostname of the node that is meant to process the import. Finally, `Iteration` defines the current iteration of the experiment.These parameters are used by `prepare_load` to build the SQL statements that are used for the data import and stored in the database. The experiment is then started by using a Linux script that queries the database for the generated import statements and submits them to the HANA instance via the command line tool `hdbsql`. Runtime information, stored in the columns "duration" and "insertspersecond" is updated after each finished import.

3.2 System Environment

According to the architecture described in section 2, a corresponding environment based on the Fujitsu FlexFrame®Orchestrator [3] is used for conducting the experiments. This environment consists of three single SAP HANA database instances (referred to as *HD1*, *HD2* and *HD3*). In all systems, the used database version is 1.00.63.381310. Each of the instances is hosted on a dedicated server node of the type Fujitsu PRIMERGY RX600S6 with each four Intel Xeon E7-4870 processors and 1 TiB of main memory. These nodes are redundantly connected to a central storage system via a 10 Gbit Ethernet and two redundant Cisco N5548 switches. The central storage system consists of a NetApp FAS 3250 HA with two NetApp disk shelves DS2246. These provide a total storage of 28 TB. Although the experiment is designed to work on different hosts, only one system (*HD1*) is utilized

in the course of the experiments in order to avoid interdependencies between the different servers since these access the same storage.

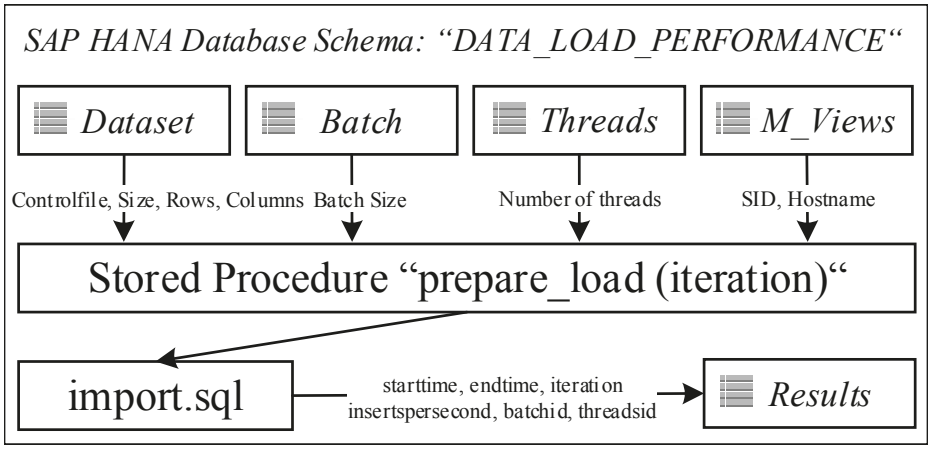

Figure 3: Components for Automating the Experiments

3.3 Datasets

Within the context of the experiments, three different datasets are used to create an initial knowledge base that can be used for parameterizing future imports. The datasets comprise of performance information that was monitored within productively running system environments. Hence, the datasets represent a realistic case. The characteristics of the respective datasets are summarized in Table 1.

Table 1: Characteristics of the Datasets used in the Course of the Experiments

	Size in GB	Number of Columns	Number of Rows
Dataset D_1	3.52	9	74,620,735
Dataset D_2	9.24	12	135,711,680
Dataset D_3	18.1	13	233,321,789

In each dataset, the used data types are `varchar`, `integer` and `double`. The dataset D_1 consists of disk I/O statistics for thousands of different servers, which host various types of SAP systems. Dataset D_2 includes statistics for whole SAP systems, while dataset D_3 comprises of statistics for single instances within a system. Each of the datasets is stored in a single CSV file and corresponds to a single table in the database.

3.4 Experiment Runs

In order to build the knowledge base, three experiments – one for each dataset – are performed in different iterations. The parameters and the outcomes of the experiments are stored in the schema as illustrated in Figure 2. Within the single iterations, the configuration parameters ("Threads" and "Batch") are varied in order to investigate a vast amount of combinations of the parameters. Using the bulk load feature for importing files into a HANA database, the number of threads can be varied between 1 and 256 [9]. For the batch size, no limitation is stated in the SAP HANA reference. Therefore, permutations without repetitions of both parameters are built by the procedure `prepare_load`. In order to cover different values of both parameters, the procedure varies them as follows: in the first iteration, the number of threads is varied between 2 and 256 in steps of 2^n. The batch size is initially varied from 10 to 100,000,000. The first value of the batch size is ten, while the following values are determined by multiplying the previous value by ten. Therefore, 64 combinations of "Threads" and "Batch" are computed in the first iteration of each dataset. In the following iterations, the parameter combinations of the top results concerning the achieved inserts per second are selected and used for determining the parameters for the next iteration. In order to define a stop criterion, the top 64 results are initially selected for determining the new parameters in the first iteration. In each of the following iterations, the number of top results is bisected (top 32, top 16, etc.) until only one result remains in the last iteration.

Table 2: Identification of new Parameters

Batch Size	Threads (Iteration n)	Inserts per Seconds	Threads (Iteration n+1)	Inserts per Second
100	32	666,256	48	58
100	64	672,258		
1,000	128	888,342	—	—
10,000	256	1,081,459	—	—
...	...	...	...	...

For extending the knowledge base and in order to avoid computing the same parameters several times within an experiment, only new combinations of input parameters are used in a new iteration. This is ensured by computing new parameter combinations, using the top results from the previous iterations as a basis. In order to get new parameter combinations, the mean values between two adjacent values are determined and used as new values for a permutation as shown in Table 2. Parameter combinations that have already been investigated in a previous iteration are excluded from the permutations. However, the mean values are only determined if the adjacent parameters belong to the associated parameter. For this

reason, in Table 2, no mean value is computed for the thread size "128" in line 3: it does not correspond to the same batch size as the other values for "Threads".

Parameter combinations that have not been investigated yet are used in the next iteration by inserting these values in the configuration tables. The respective dataset is then imported into the defined table by using the new parameters. Before each import, the table is dropped and recreated in order to ensure that the formerly imported data has no effect on the performance of the current data import. By applying this method, the knowledge base was extended by each iteration up to a total amount of 878 available reference imports.

3.5 Results

We performed import experiments for three different datasets (confer Table 1) with a total of 20 iterations in order to successively approximate the respective most suitable parameter combination. In total, 878 data imports were performed. The results were analyzed using analytical capabilities of SAP HANA and are illustrated in the following.

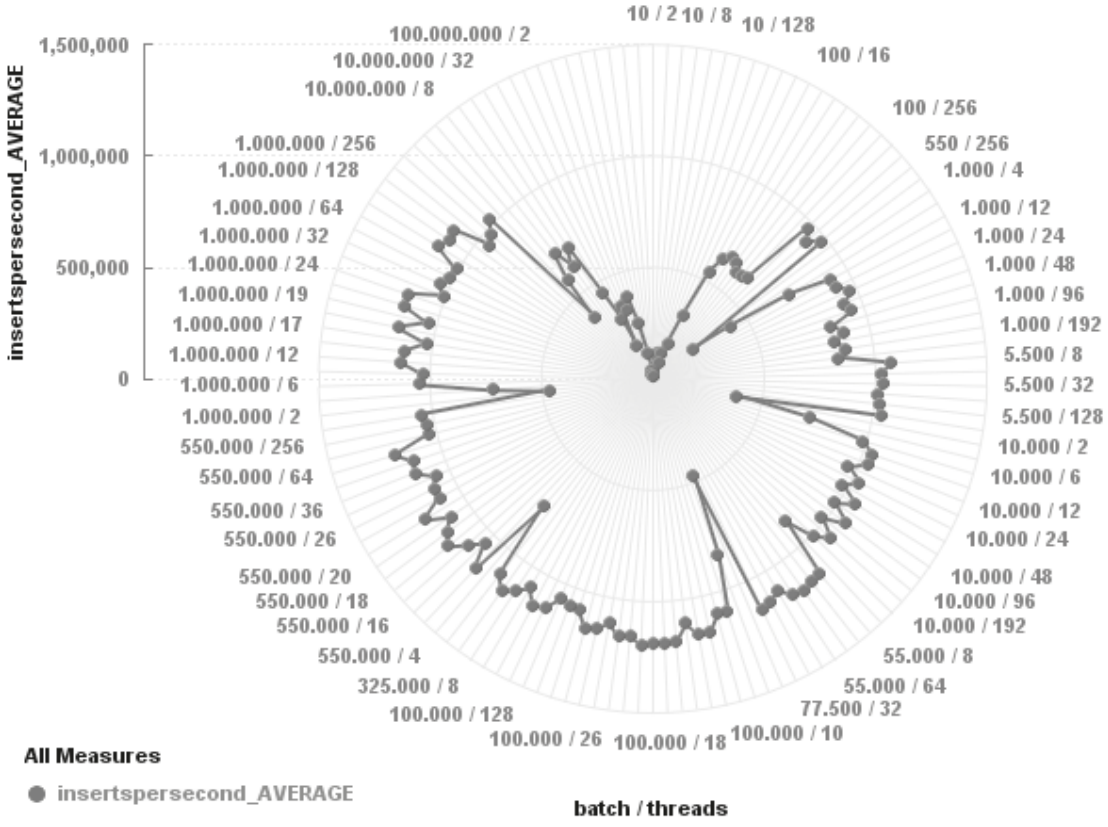

Figure 4: Avarage Results for Dataset D_1

For dataset D_1 we ascertained an average of 808,202.392 inserts per second across all performed parameter permutations and a maximum value of 1,203,560.242 inserts per second by using 64 threads and a batch size of 550,000. Figure 4 shows the results for all performed imports of dataset D_1. Since the results are sorted by batch size, the spectrum shows that batch sizes lower than 100 and greater than 1,000,000 led to comparatively poor performance.

In contrast to that, dataset D_2 was imported with an average performance of 568,983.442 inserts per second. An import applying 38 threads and a batch size of 3,025 led to the shortest duration and resulted in 789,021.395 inserts per second. We excluded the corresponding charts for Dataset D_2 and D_3 since the imports show a behavior similar to dataset D_1. Dataset D_3 has been imported with a maximum performance of 911,413.238 inserts per second, using a batch size of 100,000 and 20 threads as well as 24 threads. In average, dataset D_3 was imported with 688,486.184 inserts per second, which implies greater performance than dataset D_2, but less than dataset D_1. The gap between the greatest and smallest batch size, recognizable on top of Figure 4, became smaller for Dataset D_3, leading to the assumption that high batch sizes are more suitable for large tables.

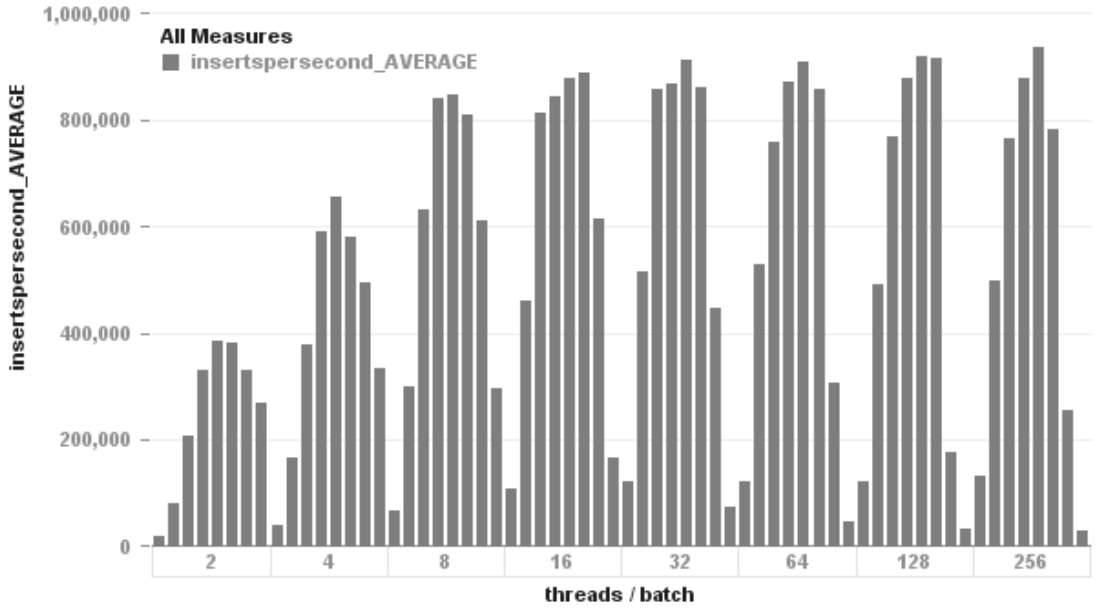

Figure 5: Avarage Results over all Datasets

Independently of the size of a dataset, low numbers of threads led to comparably poor import performance. These imports are represented by the data points closest to the center inside the corresponding Figure 4. This correlation is also confirmed by Figure 5, which compares the import performances of the first iteration across all datasets.

Therefor, Figure 5 illustrates the results for each labeled number of threads combined with the following batch sizes: 10; 100; 1,000; 10,000; 100,000; 1,000,000; 10,000,000 and 100,000,000. Any configuration including four or less threads performed significantly slower than any other number of threads between 8 and 256. Within each thread interval, the averagely best performing batch sizes ranged from 10,000 to 1,000,000.

Comparing the imports parameterized by the experimentally determined values with both the default import configuration (no parameters passed) and a configuration recommended by SAP (10; 10, 000) [9], we ascertain a significant difference between imports performed with the default values of the parameters and the imports using the parameter values that were experimentally determined as illustrated in Figure 6.

The parameter configuration recommended by SAP led to a number of inserted rows per second that is located between default and optima with a lag varying from 93,064 (D_2) to 152,563 (D_1) inserts per second compared to the respective parameters that were determined in the context of the experiments.

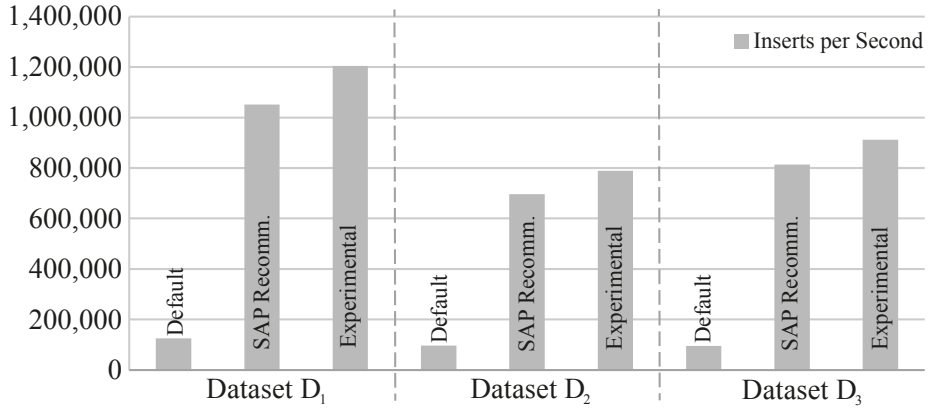

Figure 6: Comparison of the Results for each Dataset

4 Self-Configuring Data Imports

Rather than importing datasets by using the default or predefined parameter values or by randomly choosing the parameters for the import, the import of new datasets could be conducted based on similar data that was previously loaded. In order to be able to determine the similarity of datasets, a knowledge base was built as described in the previous section. The data generated by means of the experiments can now be used in order to determine suitable parameter values for an import based on historical data. Therefore, it is first of all necessary to compute the similarity of all previously imported datasets in relation to the dataset that is to be imported.

The similarity of data can be computed in various ways, for example by applying kernel functions [8] or by using distance functions such as presented in [2]. In this paper, a distance function was chosen to compute the similarity of two datasets, the canberra distance [5], which is defined in Equation 1.

$$d(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{n} \frac{|p_i - q_i|}{|p_i| + |q_i|} \tag{1}$$

$\mathbf{p}$ and $\mathbf{q}$ are vectors that represent a particular dataset and comprise the parameters of a dataset: its size, its number of rows and its number of columns. Respectively, p_i and q_i are elements within the vectors. The most similar dataset in relation to the dataset that is to be imported is the one with the smallest distance. The distance is determined for all datasets that were previously imported and can hence be used to query for the parameters that have led to the best import result with regard to the most similar dataset.

But up to this point, a suitable configuration for the import parameters would only be determined based on historical data. However, in a cloud environment, it is unlikely that each system is idle due to the fact that a provider may serve a vast amount of users [1]. Thus, it is reasonable to assume that the individual hosts are differently utilized. Hence, the determined configuration of input parameters may not be applicable due to the current utilization of a host. For example, user A may upload data that results in an import configuration of (256; 100,000) according to the knowledge base. At the same time, user B works on the system, consequently the current utilization is very high and the additional import would overload the node. This should be avoided due to possible contractual penalties as a result of the violation of SLAs. Therefore, it would not be possible to employ the optimal import configuration for user A due to the current state of the system. It would rather be desirable to adapt the configuration according to the current state of the system.

In order to cope with such a scenario, a solution was implemented that determines a suitable import configuration based on the historical data and adapts this configuration according to the current utilization (in terms of the CPU utilization) of the node. In detail, this includes the following steps:

1. Find a node (refered to as "target node") that provides enough free main memory to process the import using a first fit approach.
2. Determine a suitable parameter configuration. Therefore, a procedure named `getConfig` is called via `hdbsql` on the target node. The relevant parameters (CPU utilization, size of dataset, number of rows and columns) are identified at the OS-layer and passed to the procedure.
    a) The canberra distance is computed for all datasets in the database (cf. Equation 1).
    b) Based on the distance, the best parameter combination is then determined by using the "Results"-table.

c) The number of threads determined in the previous step is then linearly adjusted to the current CPU utilization of the node.

d) Finally, the corresponding import statements are generated and returned by the procedure.

3. The returned import statements are stored in an SQL file that is parsed and then passed to `hdbsql` in order to start the import.

This implementation was tested for an additional dataset D_4 whose characteristics are listed in Table 3.

Table 3: Characteristics of the Dataset used to Test the Self-Configuring Import

	Size in GB	Number of Columns	Number of Rows
Dataset D_4	20.65	11	265,046,669

The dataset was processed as described before. For D_4, the most similar dataset is D_3 with $d(D_4, D_3) = 0.064$. Therefore, the parameter-combination that was determined by the procedure is (24; 100,000). This combination was applicable since there were no other active users on the system so that and no adjustment of the parameters had to be conducted. In addition to this import, the dataset was also imported by using the default values for threads and batch and the values suggested in [9]. Figure 7 illustrates the corresponding results.

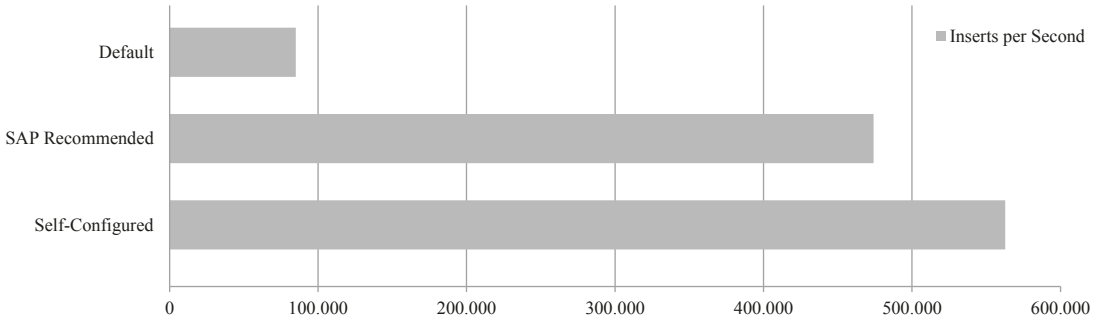

Figure 7: Comparison of the Results for Dataset D_4

As can be derived from Figure 7, the import that was determined by our implementation achieved the highest number of inserts per second and hence was the best performing import. In comparison to the import with the default parameters, the import speed improved by 84.89 %. With regard to the parameter values suggested in [9], an improvement of 15.47 % was achieved. This demonstrates the ability of the implemented approach to both automate and improve the speed of

data imports in SAP HANA cloud environments. Up to now, only a simple approach has been implemented for the proof-of-concept. Certainly there are many approaches that could be applied in order to determine suitable import parameters and to adjust these to the current state of the system. For example, more advanced task scheduling algorithms could be used in order to assign imports to suitable nodes. Additionally, this approach may be combined with a fair queuing approach in order to guarantee a minimal level of service to all users in terms of resources that can be allocated for the import.

5 Conclusion

In this contribution, a concept for self-configuring data imports in SAP HANA cloud environments was designed and implemented. It can be used to automatically determine a suitable combination of import parameters for a specific dataset. Furthermore, the developed concept takes into account the current utilization of the system and correspondingly adjusts the import parameters in order to prevent a system from being overloaded. Accordingly, an architecture that provides the features required for such a solution was designed and verified in this contribution. In the developed concept, the parameterization of an import that is to be initiated bases on the results of similar datasets that were previously imported. Since historical data is needed be able to identify similar datasets and their respective import parameters, different experiments were conducted with the aim to generate historical data. In the course of these experiments, 878 imports with a total size of about 34 TB of data were automatically performed. This data can be used to identify suitable parameters for a new import. In order to verify the developed concept, it was tested with an additional dataset. This additional import suggests that a generic import configuration that aims to result in the shortest import durations does not exist for a particular SAP HANA database server. Rather, the parameters seem to depend on the characteristics of the datasets, such as their size and their number of rows. Hence, by applying the concept developed in this paper, the import speed was improved by 15 % as compared to a parameterization suggested by SAP, and by 85 % compared with the default parameterization. Therefore, building a knowledge base that comprises of configurations and performance results of the imports can improve further imports by selecting appropriate parameters based on similar configurations that were previously applied. However, there are still aspects that need to be addressed in future research. Further improvements may address the process of determining similar datasets. We expect a different prioritization of the individual parameters when investigating their respective impacts on the performance of imports which might thus lead to a surpass of the currently applied canberra distance. Additionally, other aspects of datasets, such as the maximum width of a column, may affect the speed of imports and hence may need to be included when computing the similarity of datasets. Furthermore, we will extend the experiment by including differently sized hosts in order to assess the dependence of the import parameters on particular hardware resources.

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". In: *Commun. ACM* 53.4 (Apr. 2010), pages 50–58.

[2] M.-M. Deza and E. Deza. *Dictionary of distances*. Elsevier, 2006.

[3] *FlexFrame Orchestrator Version 1.0A*. Technical White Paper. Fujitsu Technology Solutions GmbH. 2014.

[4] A. R. Hevner, S. T. March, J. Park, and S. Ram. "Design Science in Information Systems Research". In: *MIS Q.* 28.1 (Mar. 2004), pages 75–105.

[5] G. Lance and W. Williams. "Mixed-Data Classificatory Programs I – Agglomerative Systems". In: *Australian Computer Journal* 1.1 (1967), pages 15–20.

[6] D. Laney. *3D Data Management: Controlling Data Volume, Velocity, and Variety*. Technical report. 208 Harbor Drive, PO Box 120061, Stamford: META Group, Feb. 2001.

[7] S. LaValle, E. Lesser, R. Shockley, M. S. Hopkins, and N. Kruschwitz. "Big data, analytics and the path from insights to value". In: *MIT Sloan Management Review* 53.2 (2011), pages 20–31.

[8] K. Rieck, P. Laskov, and K.-R. Müller. "Efficient Algorithms for Similarity Measures over Sequential Data: A Look Beyond Kernels". In: *Pattern Recognition*. Edited by K. Franke, K.-R. Müller, B. Nickolay, and R. Schäfer. Volume 4174. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pages 374–383.

[9] *SAP HANA SQL and System Views Reference: IMPORT FROM*. SAP SE. 2014. URL: http://help.sap.com/saphelp_hanaplatform/helpdata/en/20/f712e175191014907393741fadcb97/content.htm (visited on 2015-03-03).

[10] *SAP HANA Tailored Data Center Integration – Frequently Asked Questions*. SAP SE. 2014. URL: http://www.saphana.com/docs/DOC-3634 (visited on 2015-03-03).

[11] D. Talia. "Clouds for Scalable Big Data Analytics". In: *Computer* 46.5 (May 2013), pages 98–101.

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
93	978-3-86956-318-3	**ecoControl : Entwurf und Implementierung einer Software zur Optimierung heterogener Energiesysteme in Mehrfamilienhäusern**	Eva-Maria Herbst, Fabian Maschler, Fabio Niephaus, Max Reimann, Julia Steier, Tim Felgentreff, Jens Lincke, Marcel Taeumel, Robert Hirschfeld, Carsten Witt
92	978-3-86956-317-6	**Development of AUTOSAR standard documents at Carmeq GmbH**	Regina Hebig, Holger Giese, Kimon Batoulis, Philipp Langer, Armin Zamani Farahani, Gary Yao, Mychajlo Wolowyk
91	978-3-86956-303-9	**Weak conformance between process models and synchronized object life cycles**	Andreas Meyer, Mathias Weske
90	978-3-86956-296-4	**Embedded Operating System Projects**	Uwe Hentschel, Daniel Richter, Andreas Polze
89	978-3-86956-291-9	**openHPI: 哈索•普拉特纳研究院的MOOC（大规模公开在线课）计划**	Christoph Meinel, Christian Willems
88	978-3-86956-282-7	**HPI Future SOC Lab : Proceedings 2013**	Christoph Meinel, Andreas Polze, Gerhard Oswald, Rolf Strotmann, Ulrich Seibold, Bernhard Schulzki (Hrsg.)
87	978-3-86956-281-0	**Cloud Security Mechanisms**	Christian Neuhaus, Andreas Polze (Hrsg.)
86	978-3-86956-280-3	**Batch Regions**	Luise Pufahl, Andreas Meyer, Mathias Weske
85	978-3-86956-276-6	**HPI Future SOC Lab: Proceedings 2012**	Christoph Meinel, Andreas Polze, Gerhard Oswald, Rolf Strotmann, Ulrich Seibold, Bernhard Schulzki (Hrsg.)
84	978-3-86956-274-2	**Anbieter von Cloud Speicherdiensten im Überblick**	Christoph Meinel, Maxim Schnjakin, Tobias Metzke, Markus Freitag
83	978-3-86956-273-5	**Proceedings of the 7th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering**	Christoph Meinel, Hasso Plattner, Jürgen Döllner, Mathias Weske, Andreas Polze, Robert Hirschfeld, Felix Naumann, Holger Giese, Patrick Baudisch (Hrsg.)

ISBN 978-3-86956-319-0
ISSN 1613-5652