CloudGenius: A Hybrid Decision Support Method for Automating the Migration of Web Application Clusters to Public Clouds

Michael Menzel, *Member, IEEE*, Rajiv Ranjan, *Member, IEEE*, Lizhe Wang, *Senior Member, IEEE*, Samee U. Khan, *Senior Member, IEEE*, and Jinjun Chen, *Senior Member, IEEE*

Abstract—With the increase in cloud service providers, and the increasing number of compute services offered, a migration of information systems to the cloud demands selecting the best mix of compute services and virtual machine (VM) images from an abundance of possibilities. Therefore, a migration process for web applications has to automate evaluation and, in doing so, ensure that Quality of Service (QoS) requirements are met, while satisfying conflicting selection criteria like throughput and cost. When selecting compute services for multiple connected software components, web application engineers must consider heterogeneous sets of criteria and complex dependencies across multiple layers, which is impossible to resolve manually. The previously proposed CloudGenius framework has proven its capability to support migrations of single-component web applications. In this paper, we expand on the additional complexity of facilitating migration support for multi-component web applications. In particular, we present an evolutionary migration process for web application clusters distributed over multiple locations, and clearly identify the most important criteria relevant to the selection problem. Moreover, we present a multi-criteria-based selection algorithm based on Analytic Hierarchy Process (AHP). Because the solution space grows exponentially, we developed a Genetic Algorithm (GA)-based approach to cope with computational complexities in a growing cloud market. Furthermore, a use case example proofs CloudGenius' applicability. To conduct experiments, we implemented CumulusGenius, a prototype of the selection algorithm and the GA deployable on hadoop clusters. Experiments with CumulusGenius give insights on time complexities and the quality of the GA.

Index Terms—Cloud migration, migration process, selection problem, criteria set, decision-making, decision support

1 Introduction

A web application is a computer software application, which interacts with users through a frontend programmed using browser-based language (such as JavaScript and HTML). Web applications are typically accessed by million of users over the internet via a common web browser software (e.g., Internet explorer, Firefox, etc.). Common web applications include webmail, online retail sales, online auctions, wikis and the like.

1.1 Motivation and the Research Problem

In the traditional web application hosting model [1], hardware needs to be provisioned for handling peak load. However, uncertain traffic periods and unexpected variations in workload patterns may result in low utilization rates of expensive hardware. Therefore, the traditional approach of provisioning for peak workloads leads to unused or wasted computing cycles when traffic is low. With the advent of cloud computing, it is expected that more and more web applications will be hosted using cloud-based, virtualized services. Cloud computing [2] provides an elastic Information Communication Technology (ICT) infrastructure for the most demanding and dynamic web applications. Clouds provide an infrastructure (if optimally selected and allocated) that can match ICT cost with workload patterns in real-time. Cloud[1] service types can be abstracted into three layers: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [3].

Cloud computing is a disruptive technology and an adoption brings along risks and obstacles. Risks can turn into effective problems or disadvantages for organizations that may decide to move web applications to the cloud. Such a decision depends on many factors, from risks and costs to security issues, service level and QoS expectations. A migration from an organization-owned data center to a cloud infrastructure service implies more than few trivial steps. Steps of a migration to PaaS offerings, such as Google App Engine, would differ in several regards. The following steps outline a migration

- M. Menzel is with Research Center for Information Technology, Karlsruhe Institute of Technology, Karlsruhe, Germany. E-mail: menzel@fzi.de.
- R. Ranjan is with CSIRO Computational Informatics, Acton, Australia. E-mail: raj.ranjan@csiro.au.
- L. Wang is with the Centre for Digital Earth Sciences, Chinese Academy of Sciences, Beijing, China E-mail: lzwang@ceode.ac.cn.
- S. U. Khan is with the Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58108-6050. E-mail: samee.khan@ndsu.edu.
- J. Chen is with the Faculty of Information Technology, University of Technology Sydney, Sydney, New South Wales, Australia. E-mail: jinjun.chen@uts.edu.au.

Manuscript received 20 Aug. 2013; revised 5 Feb. 2014; accepted 5 Mar. 2014. Date of publication 6 May 2014; date of current version 8 Apr. 2015. Recommended for acceptance by Y. Xiang. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TC.2014.2317188

1. Background information on cloud computing, VM images, and meta-data available for VM and compute services is given in Supplemental Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2014.2317188.

0018-9340 © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

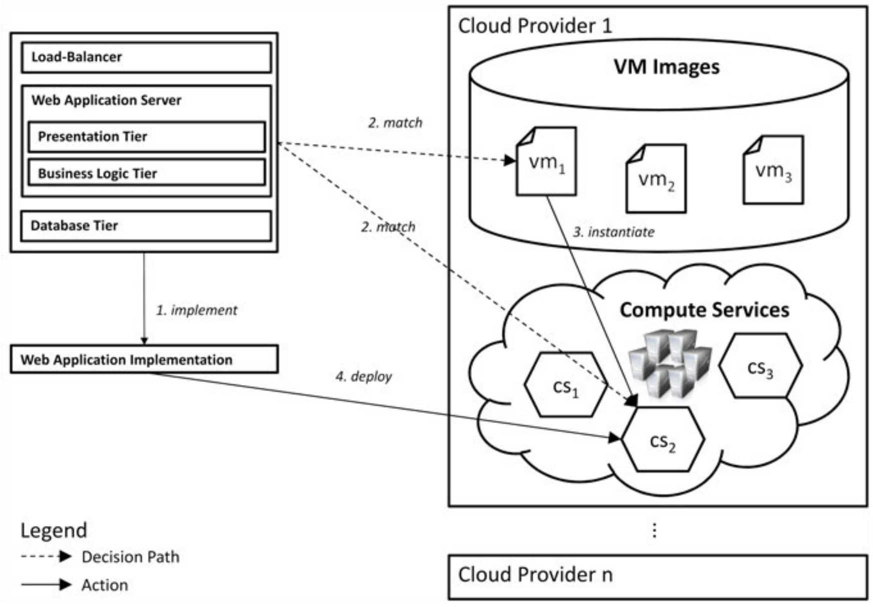

Fig. 1. Example of a VM image (PaaS) and compute (IaaS) service selection.

of an organization's web application to an equivalent on a IaaS such as Amazon Web Services (AWS) EC2, GoGrid, Rackspace, and the like.

First, an appropriate cloud infrastructure service, or IaaS offering, is selected. This demands a well-thought decision to be made that considers all relevant factors, such as price, Service Level Agreement (SLA) level, network latency, data center location, availability, and support quality. The basis of a selection are data and QoS measurements regarding each factor that describe the quality and make service options comparable. For instance, a low-end Compute service of Microsoft Azure is 30 percent more expensive than the comparable AWS EC2 Compute service, but Azure can process application workload twice as quickly.

Second, the existing web application and its execution platform, i.e., a web/application server, a load-balancer, and a database, are transferred from the local data center to the selected cloud infrastructure service. Therefore, the web application and server must be converted into a form expected by a cloud infrastructure service. Typically, in this step, the whole web application is bundled as a VM image that consists of a software stack, from operating system and software platforms to the software containing the business logic. It is often unachievable to convert an existing web application and its server directly to a VM image format compatible with a certain cloud infrastructure service. Therefore, an adequate existing VM image offered by the cloud provider can be chosen and customized. For example, one can select existing VM images provided by bitnami [4] or thecloudmarket.com [5] to cloudify an existing web application system component (e.g., application server or database). Additionally, markets for cloud VM images exist, such as the AWS marketplace [6].

Existing images vary in many aspects, such as underlying operating system, software inside the software stack, or software versions. Therefore, selecting a functionally correct VM image becomes a complex task. Besides, choosing a comprehensive VM image helps to minimize the effort of installing a software stack on a basic image. The resulting VM image should reflect the original application server and at least replace it in a sufficing manner. Next, a migration strategy needs to be defined and applied to make the transition from the local data center to the cloud infrastructure service. A migration strategy defines procedures and the course of action to transition a system and its data to the target state. In case data must be incorporated in the web application migration, all data on the original machine must be transferred to the new system in the cloud. Moreover, all configurations and settings must be applied on the new web server in the cloud to finish creating an appropriate equivalent.

Optimal web application server QoS in cloud environments demands appropriate configuration for both VM images and cloud infrastructure services. However, no detailed comprehensive cost, as well as performance or feature comparison of cloud services exists. The key problem in mapping web application server components to cloud data centers, as depicted in Fig. 1, is selecting the best collection of VM images and compute services to ensure that a system's QoS targets are met. Furthermore, another challenge is to satisfy conflicting selection criteria related to software (e.g., operating system and popularity) and compute services (e.g., latency, cost, data center location, and so on). Additionally, components might be placed at different locations or providers to prevent outages and generate costs for the Internet connectivity.

1.2 Overview of Methods and Contributions

To address the complexities when migrating multi-component web application server clusters, we expand the migration framework CloudGenius. The CloudGenius framework [7] translates cloud service selection steps into multi-criteria decision-making problems using $(MC^2)^2$ and the Analytic Hierarchy Process (AHP) [8]. The framework, furthermore, determines the most viable VM images and compatible compute services at IaaS layer. CloudGenius originally provides a framework that guides through a cloud migration process and offers a model and method to determine the best combined and compatible choice of VM images and compute services for a single web server. With enhancements to the framework, we provide comprehensive support

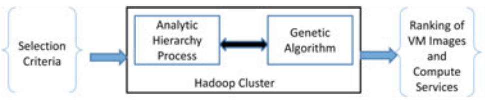
Selection Criteria
Analytic Hierarchy Process
Genetic Algorithm
Hadoop Cluster
Ranking of VM Images and Compute Services

Fig. 2. Overview of hybrid multi-goal optimization heuristic method (HMOHM) approach.

for migrations of distributed, multi-component (web/application server, database, and load-balancer) web application clusters while factoring in data flow dependencies of components raising network traffic costs.

As the solution space for the problem grows exponentially, we developed a Hadoop and Genetic Algorithm (GA)-based approach to cope with computational complexities in a growing market of cloud service offerings. By combining a GA with AHP, we created a Hybrid Multi-Goal Optimization Heuristic Method (HMOHM). Details on the HMOHM can be found in Fig. 2. A new challenge resulting from the combination of GA and AHP is to transfer subjective opinions stated once into an AHP-based fitness function. Therefore, we developed a novel approach for AHP-based fitness functions in GAs. AHP requires pair-wise comparisons among all alternatives to normalize values for an absolute scale. This becomes unsolvable with a potentially infinite number of alternatives considered in a GA. Therefore, we needed to develop a novel way for speeding up the execution time for HMOHM. To this end, we implemented a parallel version of the HMOHM over hadoop clusters. Specifically, the main contributions of this paper are:

- We clearly identify the most important selection criteria, selection goals, and cloud service alternatives, considering the use case of migrating a web application cluster to public cloud services such as Amazon EC2 and GoGrid.
- We extend analytical formulations and models of our previous work [7] for handling the migration of web server cluster components across multiple cloud data centers spanning over geographically distributed network boundaries.
- A hybrid decision making technique is proposed that combines multi-criteria decision making (AHP) and evolutionary optimization techniques (genetic algorithms) for selecting best compute service and VM image.
- A comprehensive experimental evaluation is carried out based on a realistic scenario for verifying the performance of the proposed decision making technique.

The paper is structured as follows. First, we discuss related work in Section 2. Then, the extended CloudGenius framework is presented in Section 3. The framework introduces a migration process and formal model which forms the basis for decision support within the process. Afterwards, we present CumulusGenius, a prototypical implementation of the framework in Section 4. In Section 5, we present a use case and the results of experiments on the time complexity and search space of CumulusGenius. We conclude and discuss future work in Section 6.

2 Related Work

Multi-component web services have been introduced and defined by the web service community [9]. Multi-component setups in the cloud are described in the CAFE framework [10] and in TOSCA [11].

In the context of decision-making and cloud computing, a range of approaches apply service matching using requirements from service level agreements [12], [13], [14], and [15]. Other methods employ decision-making methods. Saripalli and Pingali [16], Li et al. [17], and Han et al. [18] proposed multi-criteria decision-making methods to evaluate decision scenarios in the cloud context, including cloud services and providers. Rehman et al. [19] gave an overview of multi-criteria approaches in the cloud context. Moreover, Chan and Chieu [20] provided service and provider evaluation methods which lack multi-component support. Dastjerdi et al. [21] considered virtual machine images in a matchmaking-based approach.

Multiple approaches for multi-component setups in the cloud have applied optimization techniques [22], [23], [24], and [25] for selecting hardware resources as a cloud provider. In contrast, performance measurement techniques [17] compare cloud infrastructure services to quantitative criteria (throughput, etc.) from a customer side. However, the need to consider VM images has largely been ignored. Also, existing approaches are missing a migration process with transparent decision support and adaptability to custom criteria.

Khajeh-Hosseini et al. [26] developed the Cloud Adoption Toolkit that offers a high-level decision support for IT system to clouds with the focus on risk management and a workload cost model. While we apply AHP to select optimal combination of web server images (PaaS) and corresponding compute services (IaaS), Godse and Mulik [27] applied AHP for selecting optimal SaaS product.

Zheng et al. [28] proposed an approach to cloud services' selection based on similarity of concepts in parameters based on WordNet. On the other hand, Kang and Sim [29] and Zhang et al. [30] uses ontology for semantic similarity based cloud service search and reasoning. While these approaches aids in understanding of the configuration of cloud services, they do not offer any support for automating the process of migrating web application services to public clouds.

To the best of our knowledge, we are first to propose a hybrid decision making technique that combines evolutionary optimization methods, multi-criteria decision making methods (AHP), and massively parallel processing Hadoop programming model to enable fast, optimized and flexible selection of cloud VM images and corresponding infrastructure services.

3 CloudGenius for Web Applications

To additionally support migrations of web applications with a compute cluster architecture (referred to as web application clusters) to cloud infrastructures, we propose an extension of the CloudGenius framework. One feature of the extended framework is the evolutionary cloud migration process model. The process model integrates existing migration approaches and methods to support multi-criteria-based decisions to select cloud VM images and compute services for multiple components. In the following sections, we present the process model for multi-component migrations of web application clusters

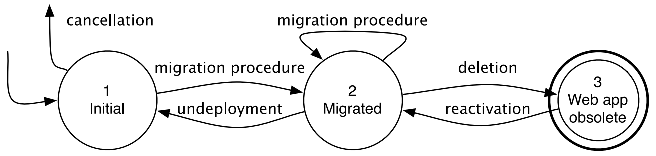

Fig. 3. States in an evolutionary cloud migration.

and give details on the formal model. Moreover, we point out required user input and flexibilities, and present methods to select VM images and services from the abundance of offerings. Finally, we give insights into the evaluation of clusters considering network costs and constraints, and look into computational complexities.

3.1 Evolutionary Cloud Migration Process

Migrations of web applications to the cloud are expected to be linear transitions from the state of "not migrated" to "migrated". However, web applications and related software stacks introduce complexity. Therefore, migrations should involve multiple repetitions and reconsiderations of past actions. CloudGenius accommodates the vicissitudes within a cloud migration by embedding decision support methods into an evolutionary migration process model. Also, existing migration strategies can be employed within the process model.

CloudGenius' migration process model describes the states of a web application as depicted in a finite state machine in Fig. 3. From an initial state a web application is migrated by an application engineer into the cloud with an optional cancellation path. From its migrated state a web application can be migrated within the cloud, be it between providers or services, or due to changes in the software stack. Eventually, a web application might be replaced or needs to be disposed and moves to an inactive state. CloudGenius provides decision support in the transition phases to a migrated web application, except reactivation. The migration process model describes the steps within a transition and embeds the decision support.

Fig. 4 depicts CloudGenius' evolutionary migration process model for clusters of web applications in Business Process Model and Notation (BPMN) 2.0. The process is divided into two lanes: (1) "user input" lane representing application engineers and domain experts and (2) "CloudGenius" lane which represents an implementation of the framework.

The process begins with an initial decision between a cloud and non-cloud infrastructure. Subsequently, an engineer states preferences and requirements, and lets CloudGenius recommend a list of feasible cluster solutions. Every solution comprises a mapping of a cluster component to a target platform constituted by a VM image and a compute service. In case no satisfying solution has been identified, the process allows to end an infeasible cloud migration. Otherwise, the process continues with migration steps still offering the chance to return to an earlier stage in the process. Thereby, CloudGenius becomes an evolutionary approach that facilitates cycles in a migration. In case of an user-initiated abort in the process, an intentional reset to the fundamental cloud decision activity (cloud versus non-cloud) is forced. This gives an engineer the chance to reconsider the fundamental infrastructure decision or to skip forward and modify retained preference and requirement statements.

Eventually, an engineer ends up with a successfully migrated web application cluster. In case of discontinuation of the process, a validation that cloud infrastructures are yet an unsatisfying choice has been attained. In conclusion, the process model supports an evolutionary approach to a migration. The evolutionary nature is facilitated through the chances for reconsiderations by returning to earlier steps within the process or by reapplying the whole process model (see Fig. 3).

3.2 Formal Model of CloudGenius

CloudGenius' formal model is extended with the notion of components (C), compatibilities (D, E, F) and network traffic (N_{in}, N_{out}) to facilitate clusters. Table 1 summarizes all parameters of the original and the extended model.

The extended formal model of CloudGenius incorporates l components c_h which are part of a cluster $\check{F}$. Furthermore, the model includes m images a_i and n cloud infrastructure services s_j of o providers p_k. C is the corresponding set of software components in a cluster, A the set of VM images, S the set of cloud infrastructure services, P the set of cloud providers, and I the set of component connections. Every VM image a_i, compute service s_j and any combination thereof (x_l) owns numerical and non-numerical attributes noted in the sets $\hat{A}_{a_i}$, $\hat{A}_{s_j}$, $\hat{B}_{a_i}$ and $\hat{B}_{s_j}$. χ represents a value connected with a numerical

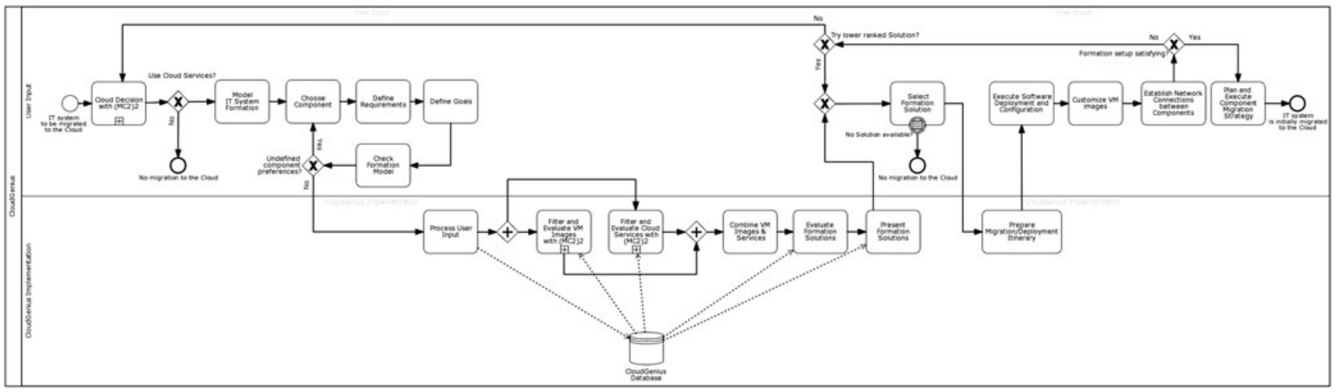
User Input
CloudGenius
CloudGenius Implementation
IT system to be migrated to the Cloud
Cloud Decision with (MC2)2
Use Cloud Services?
No
No migration to the Cloud
Model IT System Formation
Choose Component
Define Requirements
Define Goals
Check Formation Model
Undefined component preferences?
Yes
No
Process User Input
Filter and Evaluate VM Images with (MC2)2
Filter and Evaluate Cloud Services with (MC2)2
Combine VM Images & Services
Evaluate Formation Solutions
Present Formation Solutions
CloudGenius Database
No
Try lower ranked Solution?
Yes
Select Formation Solution
No Solution available?
No migration to the Cloud
Prepare Migration/Deployment Itinerary
Execute Software Deployment and Configuration
Customize VM images
Establish Network Connections between Components
Formation setup satisfying?
No
Yes
Plan and Execute Component Migration Strategy
IT system is initially migrated to the Cloud

Fig. 4. Cluster migration process of the extended CloudGenius framework.

TABLE 1
Extended Formal Model in CloudGenius

Parameter	Description
$\check{F}$	cluster to migrate consisting of components C
$C = \{c_1, ..., c_l\}$	set of l software components
$A = \{a_1, ..., a_m\}$	set of m cloud VM images
$S = \{s_1, ..., s_n\}$	set of n cloud infrastructure services
$P = \{p_1, ..., p_o\}$	set of o cloud providers
$D = \{d_1, ..., d_q\}$	set of q image-service compatibilities (a_i, s_j)
$E = \{(a_i, a_j) \| a_i, a_j \in A)\}$	set of inter-image compatibilities
$F = \{(s_i, s_j) \| s_i, s_j \in S)\}$	set of inter-service compatibilities
$I = \{(c_i, c_j) \| c_i, c_j \in C\}$	set of relations between components
$R_A = \{r_{A,1}, ..., r_{A,r_A}\}$	set of r_A cloud image requirements
$R_S = \{r_{S,1}, ..., r_{S,r_S}\}$	set of r_S cloud service requirements
$R_{c_h,X} = \{r_{X,1}, ..., r_{X,r_X}\}$	set of r_X combination requirements for c_h
τ_h	cloud image a_i, service s_j, provider p_k, or combination x_l as $\tau_h \in A \cup S \cup P \cup X$
$\hat{A}_{\tau_h} = \{\alpha_{\tau_h,1}, ..., \alpha_{\tau_h,t}\}$	set of t numerical attributes of a_i, s_j, p_k, x_l
$\hat{B}_{\tau_h} = \{\beta_{\tau_h,1}, ..., \beta_{\tau_h,u}\}$	set of u non-numerical attributes of a_i, s_j, p_k, x_l
$\chi(\alpha)$	value of numerical attribute α in CloudGenius database
$\chi(\beta)$	value of non-numerical attribute β in CloudGenius database
v_{τ_h}	value of h-th τ calculated with $(MC^2)^2$
$X_{c_h} = \{x_l = (a_i, s_j) \| a_i \in A, s_j \in S\}$	set of cloud image and service combinations for c_h
$N_{out} = \{n_{out,(c_i,c_j)} \| c_i, c_j \in C\}$	set of outgoing traffic between components (in byte)
$N_{in} = \{n_{in,(c_i,c_j)} \| c_i, c_j \in C)\}$	set of incoming traffic between components (in byte)

attribute α or non-numerical attribute β. Moreover, the model introduces $r_A + r_S + r_X$ requirements:

$$\begin{aligned} \check{F}^* = arg\ max & \left(\left(\sum_{c_h \in C} u(c_h, a_i, s_j) \right) + c(F) \right) \\ s.t. \quad & (a_{i,c_h}, s_{j,c_h}) \in D, \forall c_h \in C \\ & (a_{i,c_h}, a_{j,c'_h}) \in E, \forall c_h, c'_h \in C \\ & (s_{i,c_h}, s_{j,c'_h}) \in F, \forall c_h, c'_h \in C \\ & r = true, \forall r \in R_A \cup R_S \cup R_{c_h,X}, c_h \in C. \end{aligned} \tag{1}$$

Based on the model, CloudGenius recommends a best solution for a cluster $\check{F}$ with best combinations (a_i, s_j) in X_{c_h} for every component c_h. A best solution for a cluster has the highest value according to total benefit versus network traffic costs tradeoff evaluated in a function $c(\cdot)$ for network costs and a utility function $u(\cdot)$ that considers an engineer's preferences. In addition, a best solution needs to conforms with set D. In D viable combinations of VM images a_i deployable on compute services s_j are marked. Compatibilities between components are held in sets E and F. Compatible VM images a_i and compute services s_j are marked in E and F respectively. The sets N_{out} and N_{in} hold the expected network traffic for component relations. In sum, the problem can be expressed as an optimization problem to find $\check{F}^*$ with the highest value as in Equation (1).

3.3 Cluster Modelling

Web application clusters comprise load balancer, database server, and potentially inter-connected web and application server components. Web applications might be divided into inter-connected front-end, business logic, and data layers to allow layer-independent scaling. In practice, web servers and application servers have merged, for example the Apache Tomcat and Red Hat JBoss Application Servers. Application servers not only provide an execution environment for web applications, but also contain a web server to handle http requests. Our approach, thus, focuses on application servers in the cluster modelling.

In this paper, in regards of selection of database servers, we restrict the feature selection to VM images (e.g., Bitnami MySQL image [31], 3Tera relational database images [32], and BitNami PostgresSQL image [33]) that offer relational database functionalities. Relational database VM images include pre-configured and pre-installed traditional relational database systems, e.g., MySQL, SQL Server, and PostGres. In such database systems, the data is stored in tables that have fixed schema. SQL is used as the generic language that allows to execute projections and insert, delete, and update operations on the data. Fundamentally, relational databases have proven to be good at managing structured data, especially in the application scenario where transactional integrity (ACID properties) is a requirement. In the future work, we intend to extend the HMOHM with the ability to select non-relational database cloud services (NoSQL), such as Amazon DynamoDB, HBase, and MongoDB. Unlike relational database services, NoSQL services do not yet have support for ACID transaction principles, but rather offer weaker consistency properties. Besides, data access in NoSQL systems is typically based on predefined access primitives such as key-value pairs.

In an initial step, an application engineer has to model the application as a cluster. All components of the cluster setup must be added as elements c_h to set C and interconnections must be defined in form of component pairs in set I. The engineer and domain experts must state the expected amount of outgoing and incoming data for each component in bytes in the set N_{out} and N_{in}. Some components might be added multiple times for scaling, or with distinct requirements and goals for fault-tolerance. Furthermore, a software feature must be assigned that categorizes the component. Available feature categories are web server, relational database server, and load balancer. A feature limits the set of plausible VM images and is a mandatory requirement restraining the set of viable VM images.

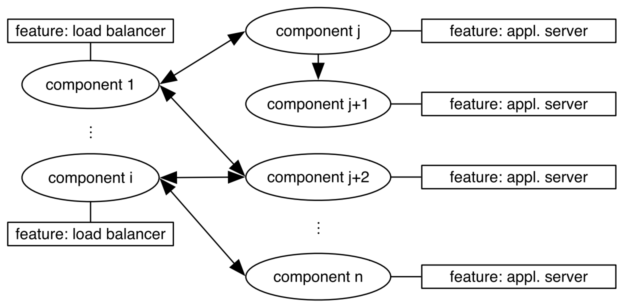

Fig. 5. Example of a cluster model.

TABLE 2
Requirement Types

Value Type	Req. Type	Boolean
Numerical	Equals	$\chi(c_i,\alpha) = v_r \vee \chi(c_j,\alpha)$
Numerical	Max	$\chi(c_i,\alpha) < v_r \vee \chi(c_j,\alpha)$
Numerical	Min	$\chi(c_i,\alpha) > v_r \vee \chi(c_j,\alpha)$
Non-numerical	Equals	$\chi(c_i,\beta) = \chi(c_j,\beta)$
Non-numerical	Not Equals	$\chi(c_i,\beta) \neq \chi(c_j,\beta)$
Non-numerical	One Of	$\chi(c_i,\beta) \in S$

The extended CloudGenius approach is capable of finding a best solution for any combination of inter-connected components. Nevertheless, we suggest a typical cluster setup depicted in Fig. 5 consisting of a set of one or more load balancers that are connected to multiple application servers which are partially inter-connected.

3.4 Software Component Requirements and Preferences

Similar to the original CloudGenius process, for each component application engineers have to formulate requirements and preferences. Requirements formulation comprises setting constraints on attributes of VM images (pertaining to web server, application server, and relational database sever) and compute services. Table 2 lists requirement filters in the $(MC^2)^2$ framework. An attribute can be required to adhere to a fixed value boundary v_r. Alternatively, the attribute of a component can be included, such as the location of c_1 must not equal the location of c_2. The table assumes $\chi(c_H)$ to be an attribute value when evaluating τ for c_h. Attributes for requirement definitions can be drawn from Tables 3 and 4 for VM images. For compute services the attributes can be seen in Tables 5 and 6, whereas for combinations the Tables 3 and 4 should be consulted. Requirements of VM image attributes need to be defined in set $R_{c_h,A}$, for services in set $R_{c_h,S}$ and for combinations in set $R_{c_h,X}$.

Given the proposed goal hierarchies depicted in Fig. 6, an engineer has to state preferences as described by the AHP. In addition to the compute service goals of the original CloudGenius framework, the attributes CPU Cores, RAM Size and Disk Size are included in the goal hierarchy. For attributes of load balancer and web server combinations (see Tables 3 and 4), two single-levelled hierarchies must be weighted.

Finally, web application engineers have to state the weight of the cloud VM image, compute service, and the combination thereof in the total value of a solution. Therefore, the weights w_a, w_s and w_{attr} need to be determined. Also, some components might be more important than others. Hence, weights for the importance of a component within the cluster have to be determined in w_{c_h}. Network cost attributes are not part of the goal hierarchies and weights are not required to determine the total network costs, given a network traffic estimation for all components. However, the importance of network costs w_T within the solution must be weighted. In parallel, the weight w_Q for the value calculated from all other criteria must be defined.

TABLE 3
VM Image Numerical Attributes

Name	Influence	Metric	Range
Hourly License Price	Negative	$/h	0-∞ $/h
Popularity	Positive	%	0-100 %
Age	Positive	Days	0-∞

TABLE 4
VM Image Non-Numerical Attributes

Name	Example Values
Virtualization Format	Xen, VMWare, ...
Operating System (OS)	Linux, Windows, ...
OS Version	Ubuntu 10.4, ...
Software Feature	Application Server, Load Balancer, Database, ...
Software	JBoss, Nginx, MySQL, ...
Software Vers.	0.8-alpha,...
Implementation Lang.	Java, Perl, Ruby, ...
Supported Impl. Lang.s	Java, Perl, Ruby, ...

TABLE 5
Compute Service Numerical Attributes

Name	Influence	Metric	Range
Hourly Price	Negative	$/h	0-∞ $/h
CPU Cores	Positive	Cores	0-∞
RAM Size	Positive	Bit	0-∞
Disk Size	Positive	Bit	0-∞
CPU Perfomance	Positive	Flops	0-∞ Flops
RAM Perfomance	Positive	Ops/s	0-∞ Ops/s
Disk Perfomance	Positive	B/s	0-∞ B/s
Max. Latency	Negative	ms	0-∞ ms
Avg. Latency	Negative	ms	0-∞ ms
Uptime	Positive	%	0-100%
Service Popularity	Positive	%	0-100%
Network Send Price	Negative	$/Byte	0-∞
Network Recieve Price	Negative	$/Byte	0-∞
Internet Send Price	Negative	$/Byte	0-∞
Internet Recieve Price	Negative	$/Byte	0-∞

3.5 Cloud VM Image and Compute Service Attributes

Numerical and non-numerical attributes for VM images are listed in Tables 3 and 4, for cloud services in Tables 5 and 6, corresponding to the proposed goal hierarchies. Attributes of VM images and compute services are applicable to filters for components of all feature categories. The selection of attributes is drawn from own observations and literature on VM images and services [21], [34], providing a basic set of attributes essential to cloud VM image and service selection. However, service attributes have a limited applicability to VM images. Therefore, we aim at gradually improving the list of attributes from usage data of a publicly available prototype [35], [36] and VM image databases, such as thecloudmarket.com [5] and BitNami [4]. Moreover, cloud compute services own

TABLE 6
Compute Service Non-Numerical Attributes

Name	Example Values
Provider	Amazon, Rackspace, ...
Location Country	Germany, Australia, ...
Architecture	32Bit, 64Bit
Owner	Amazon, Rackspace, Bitnami,...

multiple numerical attributes that imply a measurement or benchmarking. Therefore, an integration of existing approaches for costs [26], [37], performance, and latency measurements averaged over time [38] are beneficial.

In addition to the attributes of VM images and services, there exist attributes assignable to combinations of VM images and services. Attributes of combinations are in effect when a VM image is instantiated on a compute service. However, the instantiation of a VM image costs money and time. Therefore, measuring attributes of combinations is a considerable effort and filling a database with such measurement data demands investments. The number of attributes specific to instantiated VM images can become immense. For m VM images and n compute services, $m \times n$ measurements had to be made. Also, measuring instantiated VM images demands specific benchmarking tools such as (i) httperf [39] and ApacheBench for load-balancers and (ii) TPC-W [40] for web/application and relational database servers.

Consequently, we leave the choice of combination attributes to the application engineer and decision-maker. To lower the costs, the number of VM images must be decreased by setting strict requirements. The software feature requirement is a mandatory requirement. To make results comparable only VM images with similar software features should be measured. In general, CloudGenius can be extended to include all sorts of software features. However, it is noteworthy to mention that any additional feature requires specific attributes and criteria to be defined.

Nevertheless, by defining and measuring combination attributes the evaluation becomes more precise in terms of considered factors. Combination attributes are included in the evaluation function $h(\cdot)$ in Section 3.6.

3.6 Best Cluster

The original CloudGenius framework evaluates VM images, compute services, and combinations thereof with an evaluation method based on the Analytic Hierarchy Process created with the $(MC^2)^2$ framework. The evaluation method is translated into three functions that map VM images, compute services, and combinations to a value. All functions determine a normalized value according to a decision-maker's preferences and requirements. For web application clusters the original approach remains, but with the addition that evaluations are conducted per component in a cluster and, finally, are aggregated into a whole cluster value. Furthermore, network traffic needs to be considered in the value. Network traffic within a provider's data center is typically cheaper than traffic over the internet which is needed when a cluster is distributed over multiple providers.

The component-related functions $f(c_h, a_i, \hat{A}_{a_i}, \hat{B}_{a_i})$, $g(c_h, s_j, \hat{A}_{s_j}, \hat{B}_{s_j})$, and $h(c_h, a_i, s_j)$ consider component-related

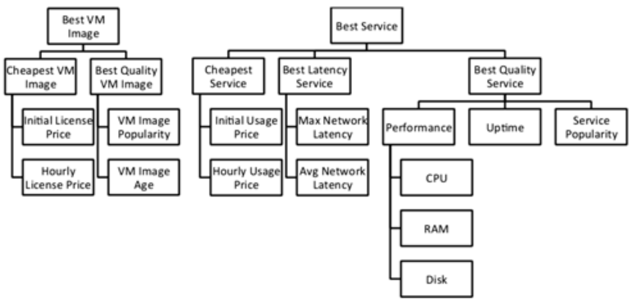

Fig. 6. Criteria hierarchies overview.

requirements $R_{c_h,A}$ and $R_{c_h,S}$ and weights (see Equations (2), (3) and (4)). Function $f(\cdot)$ returns an evaluation value for a VM image, function $g(\cdot)$ for a compute service, and function $h(\cdot)$ for every combination of a VM image and compute service. Function $i(\cdot)$ merges values from VM image, compute service, and combination attribute evaluations to a total combination evaluation value (see Equation (5)):

$$f(c_h, a_i, \hat{A}_{a_i}, \hat{B}_{a_i}) = \begin{cases} \dfrac{\sum_{j=0}^{|\hat{A}_{a_i}|} w_j \chi(\alpha_{j,a_i,+})}{\sum_{j=0}^{|\hat{A}_{a_i}|} w_j \chi(\alpha_{j,a_i,-})}, & \forall r \in R_{c_h,A} : r = \mathit{true}, \\ 0, & \text{else}, \end{cases} \\ \mapsto v_{c_h,a_i} \tag{2}$$

$$g(c_h, s_j, \hat{A}_{s_j}, \hat{B}_{s_j}) = \begin{cases} \dfrac{\sum_{i=0}^{|\hat{A}_{s_j}|} w_i \chi(\alpha_{i,s_j,+})}{\sum_{i=0}^{|\hat{A}_{s_j}|} w_i \chi(\alpha_{i,s_j,-})}, & \forall r \in R_{c_h,S} : r = \mathit{true}, \\ 0, & \text{else}, \end{cases} \\ \mapsto v_{c_h,s_j} \tag{3}$$

$$h(c_h, a_i, s_j) = \begin{cases} \dfrac{\sum_{l=0}^{|\hat{A}_{(a_i,s_j)}|} w_l \chi(\alpha_{l,(a_i,s_j),+})}{\sum_{l=0}^{|\hat{A}_{(a_i,s_j)}|} w_l \chi(\alpha_{l,(a_i,s_j),-})} & \forall r \in R_{c_h,X} : r = \mathit{true}, \\ 0, & \text{else}, \end{cases} \\ \mapsto v_{c_h,(a_i,s_j)} \tag{4}$$

$$i(c_h, a_i, s_j) = \begin{cases} w_a f(a_i, \hat{A}_{a_i}, \hat{B}_{a_i}) & \\ +w_s g(s_j, \hat{A}_{s_j}, \hat{B}_{s_j}) & \\ +w_{attr} h(c_h, a_i, s_j) & (a_i, s_j) \in D, \\ 0, & \text{else} \end{cases} \\ \mapsto v_{c_h,(a_i,s_j)}. \tag{5}$$

For component c_h, the best combination has the value $max\{v_{c_h,(a_1,s_1)}, \ldots, v_{c_h,(a_m,s_n)}\}$ calculated with $i(\cdot)$, with all values being comparable on an absolute [0, 1] scale guaranteed by normalization in the AHP. In case no alternative meets all the requirements, in a subsequent step, all alternatives that meet all but one requirement are considered. This procedure repeats until a non-empty set of alternatives is found that fulfills less requirements. Optionally, the CloudGenius process offers an opt-out to look for non-cloud options if and only if no solution satisfies the given requirements.

In a multi-component setup, the viable connections of components are defined by the sets E, F, and I. The connections included in set I must adhere to the rules from the sets E and F which define compatible VM images and services. A feasible solution of a cluster, comprising combined solution pairs (a_i, s_j) for every component, is not only the sum of the values of all contained c_h. Network costs of component inter-connections affect the quality of a solution, too. Cloud services of different cloud providers and at different locations can cause internet traffic costs. Equation (6) shows the Δ of total network traffic costs (internet and local network) for a component $c_h \in C$ connected with other components according to I. If I includes the connection (c_h, c_i) with $c_i \in C$, I avoids doubled costs and will not hold the inverse (c_i, c_h). T represents the costs for network traffic for expected communication of a multi-component cluster solution $\check{F}_i$. Let T_{c_h,c_i,R_l}, T_{c_h,c_i,S_l}, T_{c_h,c_i,R_g}, and T_{c_h,c_i,S_g} be the expected cost of incoming and outgoing local network (R_l, S_l) and internet traffic (R_g, S_g) between components c_h and c_i. These costs are calculated in advance according to sets N_{out} and N_{in}, and the providers' price scheme held in a compute service's attributes.

$$T_{\text{network},\check{F}_i} = \sum_{o \in I_{c_h}} \begin{cases} w_{T,R}\, T_{(c_h,o),R_l} & \\ +w_{T,S}\, T_{(c_h,o),S_l} & \text{provider/location equal} \\ w_{T,R}\, T_{(c_h,o),R_g} & \\ +w_{T,S}\, T_{(c_h,o),S_g} & \text{else} \end{cases} \tag{6}$$

$$j(\check{F}_i) = \begin{cases} \frac{w_Q \sum_{c_h \in C} w_{c_h} i(c_h, a_i, s_j)}{w_T\, T_{\text{network},\check{F}_i,\text{normalized}}} & \forall c_h, c_h' : (a_i, a_i') \in E \\ & \wedge (s_j, s_j') \in F \\ 0, & \text{else.} \end{cases} \tag{7}$$

$$\mapsto \mathrm{v}_{\check{F}_i}$$

All $T_{\text{network},\check{F}_i}$ are normalized to $(0,1)$ scale in AHP. Equation (7) formulates the function $j(\cdot)$ that calculates the overall value of a cluster solution instance $\check{F}_i$. The value consists of the weighted sum of combined solution values returned by function $i(\cdot)$ and the total network costs $T_{\text{network},\check{F}_i}$ of a cluster instance $\check{F}_i$. Weights reflect the importance of a component if stated by the user.

After a presentation of recommended solutions, the migration process continues with a selected cluster solution, deployment of the VM images on the compute services, and further customization and re-evaluation cycles by the engineer. With all components deployed and customized, and after following a migration strategy results in the cluster available on diverse cloud infrastructure services and distributed over data centers when stated earlier in the requirements.

3.7 Computational Complexity

The decision problem in multi-component web application migrations addressed by CloudGenius is obviously complex, creating a potentially huge search space for many VM images, services, and components. Hence, it is important to analyze the actual computational complexity of the approach to ensure its applicability. We define O of CloudGenius as following:

$$\mathrm{O}\Bigg(\underbrace{m * |\hat{B_a}| + n * |\hat{B_s}|}_{\text{requirements check}} + \underbrace{m * |\hat{A_a}| + n * |\hat{A_s}|}_{\text{images services evaluation}} + \underbrace{m * n * |D| + m * m * |E| + n * n * |F|}_{\text{feasibility check}} + \underbrace{m * n}_{\text{combined evaluation}} + \underbrace{(m * n)^l}_{\text{clusters evaluation}} \Bigg).$$

The computational complexity is proportional to the number of VM images m, services n, and components l. Computations for m images and n services comprise requirements checks in sets $\hat{B}_a$ and $\hat{B}_s$ and evaluations regarding criteria of sets $\hat{A}_a$ and $\hat{A}_s$. In addition, feasibility checks using the sets D, E, and F must be executed for all $m \times n$ combinations. The evaluation with the functions $f(\cdot)$, $g(\cdot)$, $g(\cdot)$, and $i(\cdot)$ add additional complexity. The evaluations implicate additional computation steps for AHP comprising normalization of matrices and derivation of global weights which are not included for simplicity. The formulated O shows that CloudGenius' complexity is dominated by the cluster evaluation which creates a solution space with $(m * n)^l$ clusters (Cartesian product of combinations). Therefore, the complexity is expected to grow exponentially in proportion to l, m, and n. Since CloudGenius searches in the full solution space, we are confident to find the actual best solution.

3.8 Parallel Genetic Algorithm

Meta-heuristics allow to cope with exponentially growing computational complexities. In particular, gargantuan search spaces of discrete solutions can be searched even without knowledge about the search space's structure. While meta-heuristics can cut time complexities to a fixed limit, they cannot guarantee an optimal solution. Nevertheless, the accuracy can be influenced by increasing the granted computation time. Thereby, meta-heuristics give the option to trade waiting time for accuracy. When parallelizing meta-heuristics, computation can be divided into multiple processing units which can be run on multiple server instances. This introduces the additional dimension of costs for computational resources to the tradeoff. Consequently, accuracy can be achieved by long waiting times or high investments in computational resources.

The proposed algorithm facilitates parallel computations and resembles a population-based genetic algorithm meta-heuristic due to the discrete nature of the search problem. The algorithm consists of four major steps: (1) create initial population, (2) assignment of fitness values, (3) selection of elite, and (4) evolution of a population. Step 1 only occurs once at the beginning of the algorithm, while steps 2-4 repeat until a termination criterion is fulfilled. Either a certain number of generations have been evolved or a certain amount of time has passed.

The algorithm expects a database of cloud VM images and compute services including their compatibilities as described

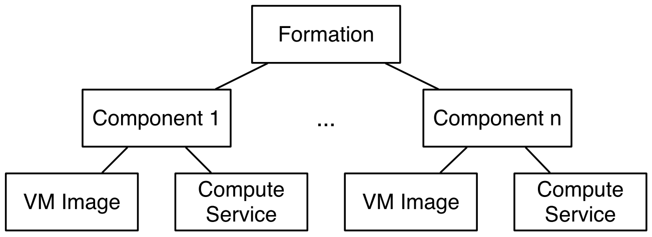

Fig. 7. Structure of a cluster solution candidate.

by CloudGenius model. Also, a cluster model, preferences, and requirements need to be defined in the model by an engineer. In addition to the model, three parameters must be set in the beginning to adjust the GA: (a) population size, (b) elite size and (c) maximum computation time.

3.8.1 Candidates and Populations

Genetic algorithms search over populations in multiple iterations referred to as generations. Every population consists of a predefined amount of solution candidates, each representing a viable and discrete solution to the problem. We propose candidates to be the cluster solutions $\check{F}_i$ as described in Section 3.6. Fig. 7 depicts the structure of a cluster solution candidate. Every candidate represents the set of component mappings to a viable VM image and compute service combination that can be evaluated with function $j(\cdot)$. To initiate the GA, an initial population must be generated. Our algorithm picks cluster solutions randomly ignoring duplicates and adds them to the population until it has grown to the expected population size (a). After this step the algorithm is ready to evolve the initial population.

3.8.2 Fitness Value of Candidate

Every population is evaluated to determine the fitness values of its candidates. Our genetic algorithm employs the evaluation function $j(\cdot)$ for each candidate $\check{F}_i$. Since the AHP determines values of candidates in a comparison matrix, the whole population must be known and evaluated together. In particular, for parallel genetic algorithms this is of special interest since every parallel evaluation task must be given the whole set of candidates in a population. For implementations of the algorithm this leads to a limitation on the population size imposed by the available memory size of tasked computation units.

3.8.3 Selection of Elite

Based on the fitness values determined in the previous step, an elite of size (b) can be drawn from the current population. The elite is represented by the set of highest ranked solution candidates in the population. Any non-elite candidates are discarded in the next generation and replaced by evolved candidates. The size of the elite has been specified as a parameter in the beginning.

With the termination of the algorithm a final elite is returned which includes a best solution candidate computed by the algorithm. The highest ranked elite candidate according to its value is the preferred cluster solution which is not guaranteed to be the globally best cluster solution in the search space.

3.8.4 Evolution of Population

While the elite candidates reside in the population, non-elite candidates are evolved to create a new generation and continue the search for best solutions. How the candidates are evolved depends on whether the algorithm applies a global or local search. Mutating candidates in only few attributes, e.g., changing the mapping of one component to a slightly different compute service or VM image, potentially leads to a local search. In contrast, substituting candidates with random cluster solutions from the search space that differ in a subset of attributes results in a global search.

To allow for global and local search we propose two evolutionary operators: (1) altering a cluster solution (mutation) and (2) substituting it (nascency). Mutating a candidate means substituting one of its component's solutions with other viable combinations of VM images and compute services. To generate a better local search, the substitution strategy should take advantage of the fact that network traffic is commonly free or cheap when staying within a provider's network. Therefore, combination alterations for one or more components should consider this common pricing model fact and choose a combination with a compute service offered by the same provider. Substitution implies choosing a random solution from the search space that is not included in the current population. Both evolutionary operators, (1) mutation and (2) nascency, are applied for each non-elite candidate with a 50 percent chance to balance global and local search.

3.8.5 Termination

Every genetic algorithm requires a termination rule that forces computations to stop and to accept the currently best solution candidate. Rules consider the current quality of the solution, or simply stop after a certain number of iterations or time limit. We propose multiple rules that lead to an eventual termination. First, the algorithm stops immediately after the sum of all uniquely discovered solution candidates passes the number of total individuals in the search space $|\{\check{F}_1, \ldots, \check{F}_n\}| = (|A| \times |S|)^{|C|}$. Therefore, a list of visited solutions helps to guarantee unique visits. Then, the excess of the search space can be determined with the ratio of search space size per population size testing $\#(\text{generations}) > \frac{(|A|\times|S|)^{|C|}}{\max\#(\text{individuals in population})}$. The algorithm halts when the number of generations exceeds the ratio. Second, a general stopping rule ends the genetic algorithm after a certain amount of time, e.g., 5 minutes, specified as parameter (c). Third, more sophisticated stopping rules can halt the algorithm when, for example, the value of the best solution has not improved over multiple generations or the actual population size undercuts the maximum population size, which means only few individuals have not been visited.

4 CUMULUSGENIUS: AN IMPLEMENTATION

With CumulusGenius [35] we provide an implementation of the model and evaluation algorithms of the framework. The CumulusGenius java library offers a data model and evaluation algorithms based on the Aotearoa AHP implementation [8] that enables the evaluation of VM images, cloud services, combinations thereof, and whole clusters. The parallel genetic algorithm described in Section 3.8 has been implemented [41] using the mahout framework for hadoop [42]. Mahout includes support for genetic algorithm implementations using the watchmaker framework [43], and helps parallelizing and making algorithms deployable on hadoop clusters [44].

A Google Web Toolkit-based web frontend with jClouds [45] integration and database of the current cloud provider landscape is currently under development [36].

5 Evaluation

5.1 Use Case

As multi-component web application system migration use case, we consider an organization's application engineer. The engineer wants to migrate the operations of its online digital store from locally managed physical infrastructure (non-virtualized) to virtualized cloud infrastructure services. The company had engineered the original application based on multi-tier architecture to decouple major functionalities across two components: (a) presentation and business logic layer with Tomcat 5.x Application servers and (b) data layer, which stores information in a relational MySQL 5.x database server. A HA Proxy 1.4.x load-balancer is employed to distribute workload across multiple application servers. To increase the dependability of the web application, the application servers shall not be placed in the same data centers or preferably at different coasts or continents.

After the organization identified cloud infrastructures as a target by applying the $(MC^2)^2$ evaluation framework in a first step, the engineer defines the web server cluster including component dependencies. The cluster comprises two Tomcat AppServer (feature application server, version > 5.0, and Ubuntu Linux) and one HA Proxy (feature load balancer with version >1.4.1). A MySQL Database Server version 5.0 is already up and running in a cloud data center. Already deployed web applications persist data to the given MySQL Database Server. The HA Proxy is connected to both Tomcats which are further connected to the MySQL database. However, the latter is not included in the cluster model. Subsequently, requirements for all components must be set such as feature, OS, and version. The two Tomcats require a different locations in order to strengthen the cluster's dependability.

The engineer states throughput and costs as high priority goals, and VM image quality more important than compute services'. Moreover, evaluation criteria for VM image and compute service preferences for both application server components and the load balancer are weighted in paiwise comparisons.

CumulusGenius owns a database with information about VM images and compute services from AWS, Terremark, and Rackspace. As no results were found for the HA Proxy, the engineer changes the version requirement to $>$ version 1.4.0 and gains cluster solutions. According to the cluster's recommendation the engineer lets CumulusGenius deploy all listed VM images to assigned compute services with jClouds. The engineer connects all running components, with Tomcat servers being placed in the US and in the EU for dependability and availability reasons. After executing a migration strategy and transferring backup data to the database server, the cluster eventually becomes available on cloud infrastructure services and the first full cycle of the migration ended.

Within weeks, the demand rises and additional Tomcat servers should be added to the cluster. The engineer reenters the process cycle repeatedly to modify the cluster. Each time, CumulusGenius suggests a new mapping, and the engineer deploys VM images accordingly and connects new deployments with existing ones. Within the evolutionary process, the engineer revises the web application deployment using new experiences and enters the CloudGenius process occasionally. In every revision, he compares his past decision with his current objectives and re-evaluates viable deployment options.

5.2 Experiments

We tested our implementation CumulusGenius in experiments on an EC2 High-Memory Quadruple Extra Large (m2.x4large) instance (eight CPU cores, 26 EC2 Compute Units, and 68.4 GB of RAM) with Ubuntu 10.04 and OpenJDK JRE 6.0 in order to analyze its solution space and the actual time complexity. To execute computations with CumulusGenius, we used whirr to deploy the experiments on a mahout-enabled hadoop cluster consisting of one master node (name node, job tracker, and mahout client) and multiple slave nodes (task tracker and data node). The parameters of the experiments are the number of VM images, services, and components. VM images and services are synthetically generated with all attributes having random, but realistic values. There is a fixed number of three providers and no requirements are defined to stress the algorithm with a full solution space. Components are randomly assigned to a provider and all inter-connected to each other. When components are offered by the same provider low network costs occur. In case of different providers, five times higher internet costs are assumed. First experiments showed an explosion in computation time for a growing number of components l, VM images m and services n. A cluster of three components and a database of five VM images and five services of three providers takes >11,725 seconds (∼325 h) to find the highest ranked of 15,625 solutions (three components with 25 viable VM image and compute service combinations each). Therefore, the implementation has been enhanced with parallel threads making it possible to exploit multiple CPU cores, and AHP's matrix normalization has been simplified as this step was identified to cause tremendous effort. The exponentially growing effort for additional components in a cluster becomes obvious with the measurements depicted in Fig. 8. The EC2 instance ran out of memory with four components and three VM images and compute services.

Since the search space is extensive, we further analyzed the quality of a simpler naive solution in 100 experiments per l, m, n variation. Instead of finding the actual best cluster, it is very cheap (∼1 ms) to neglect network traffic costs and construct a best solution from every component's best combination of VM image and compute service. Table 7 shows the average number of equal component solutions

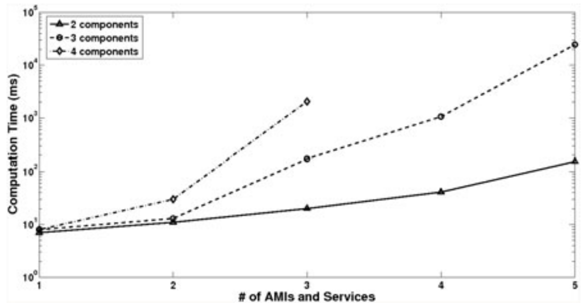

Fig. 8. Time complexities of parallel computations.

and the possibility that a naive solution equals the actual solution in all or none of the components for different parameters. The results show that for larger numbers of l, m, and n the chance for an acceptable naive solution decline and naive solutions are at best indicators for good solutions.

The genetic algorithm-based approach confirms expectations to cope with the time complexities and to provide a handle to find accurate solutions for large solution spaces depending on the invested time or money. In further experiments we compared the quality of solutions computed with the GA implementation using the mahout framework. Therefore, we compare the results with the actual best solution returned by the parallelized full evaluation (FE) that searches the whole solution space. The quality ratio as shown in Table 8 how good the GA's solution is compared to the actual solution found with a full evaluation. The GA was run with the parameters (a) population size of 100, (b) elite size of 10 and (c) eventual stopping at 5 minutes.

To test the power and quality of the GA implementation in gargantuan solution spaces, we used a five and eight node hadoop cluster (EC2 m2.x4large) and computed 10 solutions each for five components, 10,000 VM images and variations of 10 and 30 services of five providers, giving the algorithm 5 and 10 minutes time. The number of 10,000 AMIs is derived from the amount of AMIs available in the Amazon EC2 Region us-east-1 (April 2012) and gives a realistic estimate. The algorithm succeeded to find a solution, while a full non-parallelized evaluation would fail due to heap space exceedings. To get a grasp on how good the solution of the GA actually is, the naive solution serves as the only benchmark, albeit being worse than the actual solution. Table 9 sheds some light on the ratio of the GA solution versus the naive solution calculated in average of 10 runs. With more time and more compute resources the GA tends towards better solutions and beats the naive solution that omits the influence of network costs.

In summary, the experiments show that the GA approach succeeds where a full evaluation of the solution space is not possible. However, the quality of the results cannot be guaranteed to a predefined level. Increasing investments in time or additional compute resources allows to improve the quality since the chances for the genetic algorithm to find the best solution improve. Additional time allows the GA to search over more generations with same amount of compute resources, while additional resources allow to evaluate more generations in the same time. The current GA implementation beats the naive solution with more than 10 minutes of time and small clusters of five to eight nodes. Improvements in the implementation and hadoop setup might reduce computation times and increase solution quality to an yet unknown degree.

6 Conclusion and Future Work

In this paper, we presented the extended CloudGenius framework, which provides a hybrid approach that combines multi-criteria decision making technique with evolutionary optimization technique for: (i) helping application engineers with the selection of the best service mix at IaaS layer and (ii) enabling migration of web application clusters distributed across clouds.

We believe that CloudGenius framework leaves space for a range of enhancements and provides yet an amicable approach. Nevertheless, a major issue in cloud service selection is the domain of available data in the decision, i.e., completeness and freshness of a database with VM images and services, the criteria catalogs, and the quality and correctness of measured values. To address these issues, we intend to integrate cloud benchmarking approaches [38], [46] and existing databases such as CloudHarmony, bitnami, and thecloudmarket.com [4], [5], [47]. Including extended metadata information with a crawling approach [48] allows to gather more details on images and make them more differentiable, but requires additional effort to build a database.

Additionally, we plan to connect the migration process with a monitoring of the deployed system to trigger re-evaluation and decision-making in an evolutionary migration actively.

CloudGenius expects VM images to feature one component stack, such as an application server stack, instead of whole software stacks (e.g., Bitnami WordPress stack

TABLE 7
Quality of Naive Solutions versus Full Evaluation

l	m	n	AVG Equals	No Equals	All Equals
3	3	3	1.7	11%	20%
3	4	4	1.58	10%	17%
3	5	5	1.19	19%	8%
4	2	2	2.87	1%	24%
4	3	3	2.57	3%	19%

TABLE 8
Quality of GA versus Full Evaluation

l	m	n	GA vs. FE
4	2	2	100%
3	3	3	100%
3	4	4	98%
3	5	5	95%

TABLE 9
Quality of GA Solutions

time	nodes	l	m	GA vs. Naive
5	5	10	10,000	0.85
5	5	30	10,000	0.65
5	8	10	10,000	0.92
5	8	30	10,000	0.77
10	5	10	10,000	1.13
10	5	30	10,000	1.01
10	8	10	10,000	0.93
10	8	30	10,000	1.15

consisting of web server and database) or basic VM images containing an OS only. Future work should estimate customization efforts and a tradeoff. Additionally, explicit support for hybrid cloud setups, output in deployment languages, and middleware and persistence layer services will be explored in future work. It is planned to conduct an evaluation over several month with a German infrastructure provider using the CumulusGenius prototype and its web-frontend [36]. In the study not only its applicability are of interest, but also time measurements of migration process cycles.

From a web application and IT system cluster point of view, important practical challenges exist, such as (a) how to model existing web applications or IT systems as a cluster with a notation, (b) how to express and automatically handle dependencies of components, and (c) how to undertake service selection and deployment processes that consider these modelled dependencies. In future work, we aim to tackle this by adapting deployment modelling languages to consider control and data flow dependencies. We will also explore integration of existing configuration management tools such as Chef, Puppet, and whirr to CumulusGenius for configuring, deploying, and managing cloud-hosted IT system clusters.

REFERENCES

[1] Amazon Web Services. Web application hosting in the AWS cloud best practices [Online]. Available: http://media.amazonwebservices.com/AWS_Web_Hosting_Best_Practices.pdf

[2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A Berkeley view of cloud computing," EECS Dept., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2009-28, 2009.

[3] P. Mell and T. Grance, "The NIST definition of cloud computing, recommendations of the national institute of standards and technology," *NIST Special Publication*, Gaithersburg, MD, USA, 2011.

[4] BitNami. BitNami cloud images [Online]. Available: http://bitnami.org/learn_more/cloud_images

[5] The cloud market [Online]. Available: http://cloudmarket.com

[6] A. W. Services. (2013, Jan.). The AWS marketplace. https://aws.amazon.com/marketplace [Online]. Available: https://aws.amazon.com/marketplace

[7] M. Menzel and R. Ranjan, "CloudGenius: Decision support for web server cloud migration," in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 979–988.

[8] M. Menzel, M. Schönherr, and S. Tai, "$(MC^2)^2$: Criteria, requirements and a software prototype for cloud infrastructure decisions," *Softw. Practice Experience*, vol. 43, no. 11, pp. 1283–1297, Nov. 2013.

[9] R. Hamadi and B. Benatallah, "A petri net-based model for web service composition," in *Proc. 14th Australian Database Conf.-Vol. 17*, 2003, pp. 191–200.

[10] R. Mietzner, T. Unger, and F. Leymann, "Cafe: A generic configurable customizable composite cloud application framework," in *Proc. Confederated Int. Conf. OTM Conf.*, pp. 357–364, 2009.

[11] T. Binz, G. Breiter, F. Leyman, and T. Spatzier, "Portable cloud services using TOSCA," *IEEE Internet Comput.*, vol. 16, no. 3, pp. 80–85, May/Jun. 2012.

[12] V. Stantchev and C. Schröpfer, "Negotiating and enforcing QoS and SLAs in grid and cloud computing," in *Proc. Adv. Grid Pervasive Comput.*, 2009, pp. 25–35.

[13] S. Wang, Z. Zheng, and Q. Sun, "Cloud model for service selection," in *Proc. IEEE Conf. Comput. Commun. Workshop*, Apr. 2011, pp. 666–671.

[14] R. Buyya, R. Ranjan, and R. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Proc. 10th Int. Conf. Algorithms Archit. Parallel Process.*, 2010, pp. 13–31.

[15] A. Ruiz-Alvarez and M. Humphrey, "An automated approach to cloud storage service selection," in *Proc. Workshop Sci. Cloud Comput.*, 2011, pp. 39–48.

[16] P. Saripalli and G. Pingali, "MADMAC: Multiple attribute decision methodology for adoption of clouds," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2011, pp. 316–323.

[17] A. Li, X. Yang, S. Kandula, and M. Zhang, "CloudCmp: Comparing public cloud Providers," in *Proc. 10th Annu. Conf. Internet Measurement*, 2010, pp. 1–14.

[18] S.-M. Han, M. M. Hassan, C.-W. Yoon, and E.-N. Huh, "Efficient service recommendation system for cloud computing market," in *Proc. 2nd Int. Conf. Interaction Sci. Inf. Technol., Culture, Human*, 2003, pp. 839–845.

[19] Z. U. Rehman, F. K. Hussain, and O. K. Hussain, "Towards multi-criteria cloud service selection," in *Proc. 5th Int. Conf. Innovative Mobile Internet Serv. Ubiquitous Comput.*, Jun. 2011, pp. 44–48.

[20] H. Chan and T. Chieu, "Ranking and mapping of applications to cloud computing services by SVD," in *Proc. IEEE Netw. Oper. Manag. Symp. Workshops*, 2010, pp. 362–369.

[21] A. Dastjerdi, S. Tabatabaei, and R. Buyya, "An effective architecture for automated appliance management system applying ontology-based cloud discovery," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput.*, 2010, pp. 104–112.

[22] Z. Ye, X. Zhou, and A. Bouguettaya, "Genetic algorithm based QoS-aware service compositions in cloud computing," in *Proc. Database Syst. Adv. Appl.*, 2011, pp. 321–334.

[23] H. Wada, J. Suzuki, Y. Yamano, and K. Oba, "Evolutionary deployment optimization for service-oriented clouds," *Softw. Practice Exp.*, vol. 41, no. 5, pp. 469–493, 2011.

[24] H. Goudarzi and M. Pedram, "Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems," in *Proc. IEEE 4th Int. Conf. Cloud Comput.*, 2011, pp. 324–331.

[25] M. Hajjat, X. Sun, Y. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani, "Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 243–254, 2010.

[26] A. Khajeh-Hosseini, I. Sommerville, J. Bogaerts, and P. Teregowda, "Decision support tools for cloud migration in the enterprise," *IEEE Int. Conf. Cloud Comput. (CLOUD'11)*, pp. 541–548, Jul. 2011, doi: 10.1109/CLOUD.2011.59.

[27] M. Godse and S. Mulik, "An approach for selecting software-as-a-service (SAAS) Product," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2009, pp. 155–158.

[28] C. Zeng, X. Guo, W. Ou, and D. Han, "Cloud computing service composition and search based on semantic," in *Cloud Computing*, vol. 5931, M. Jaatun, G. Zhao, and C. Rong, Eds. Berlin, Germany: Springer, 2009, pp. 290–300.

[29] J. Kang and K.-M. Sim, "Ontology and search engine for cloud computing system," in *Proc. Int. Conf. Syst. Sci. Eng.*, Jun. 2011, pp. 276–281.

[30] M. Zhang, R. Ranjan, A. Haller, D. Georgakopoulos, M. Menzel, and S. Nepal, "An ontology-based system for cloud infrastructure services' discovery," in *Proc. 8th Int. Conf. Collaborative Comput. Netw., Appl. Worksharing*, 2012, pp. 524–530.

[31] BitNami. BitNami MySQL virtual machine image or cloud appliance [Online]. Available: http://wiki.bitnami.com/Components/MySQL

[32] 3Tera. 3Tera database virtual machine images or cloud appliances [Online]. Available: http://doc.3tera.com/AppLogic31/Catalog_Ref/index.htm?toc.htm?CatDataba seAppliancesMYSQLR.html

[33] BitNami. BitNami PostgreSQL virtual machine image or cloud appliance [Online]. Available: http://wiki.bitnami.com/Components/PostgreSQL

[34] S. Kalepu, S. Krishnaswamy, and S. Loke, "Verity: A QoS metric for selecting web services and providers," in *Proc. 4th Int. Conf. Web Inf. Syst. Eng. Workshops*, 2003, pp. 131–139.

[35] CumulusGenius Prototype. [Online]. Available: http://code.google.com/p/cumulusgenius/

[36] CumulusGenius Online Prototype. [Online]. Available: http://cumulusgenius.appspot.com/

[37] M. Klems, J. Nimis, and S. Tai, "Do clouds compute? A framework for estimating the value of cloud computing," in *Proc. Des. E-Business Syst. Markets, Services, Netw.*, 2009, pp. 110–123.

[38] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann, "What are you paying for? Performance benchmarking for infrastructure-as-a-service offerings," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2011, pp. 484–491.

[39] D. Mosberger and T. Jin, "httperf - A tool for measuring web server performance," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 31–37, 1998.
[40] D. Menascé, "TPC-W: A benchmark for E-commerce," *IEEE Internet Comput.*, vol. 6, no. 3, pp. 83–87, May/Jun. 2002.
[41] Mahout-based CumulusGenius Implementation [Online]. Available: https://github.com/mugglmenzel/CumulusGeniusOnMahout
[42] The Apache Mahout Project [Online]. Available: http://mahout.apache.org/
[43] Watchmaker Framework [Online]. Available: http://watchmaker.uncommons.org/
[44] The Apache Hadoop Project [Online]. Available: http://hadoop.apache.org/
[45] jClouds Multi-Cloud Library [Online]. Available: http://code.google.com/p/jclouds/
[46] S. Haak and M. Menzel, "Autonomic benchmarking for cloud infrastructures: A economic optimization model," in *Proc. 1st ACM/IEEE Workshop Auton. Comput. Economics*, 2011, pp. 27–32.
[47] CloudHarmony [Online]. Available: http://cloudharmony.com
[48] M. Menzel, M. Klems, H. A. Lê, and S. Tai, "A configuration crawler for virtual appliances in compute clouds," in *Proc. Int. Conf. Cloud Eng.*, 2013, pp. 201–209.

Michael Menzel is currently working toward the PhD degree at the Karlsruhe Institute of Technology. He is a research scientist at the Research Center for Information Technology in Karlsruhe. He has participated in multiple research projects, also with the industry, in the fields of cloud computing and cloud migration. He has published 11 scientific, peer-reviewed papers (one journal, 10 conferences). He is a member of the IEEE.

Rajiv Ranjan received the PhD degree in engineering from the University of Melbourne in 2009. He is a research scientist and a julius fellow in CSIRO Digital Productivity Flagship. Expertise in datacentre cloud computing, application provisioning, and performance optimization. He has published 110 scientific, peer-reviewed papers (seven edited books, 61 journals, 34 conferences, and 9 book chapters). He is a member of the IEEE.

Lizhe Wang received the bachelor of engineering degree with honor's (major electrical engineering and minor applied mathematics), the master of engineering degree in electrical engineering, both from Tsinghua University, P. R. China, and the doctor of engineering in applied computer science with magna cum laude from University Karlsruhe (now Karlsruhe Institute of Technology), Germany. He is a professor at Chinese Academy of Sciences. His research interests include data-intensive computing, high performance computing and Grid/Cloud computing. He is an IET fellow and a senior member of the IEEE.

Samee U. Khan received the BS degree from Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi, Pakistan, and the PhD from the University of Texas, Arlington, Texas. Currently, he is an assistant professor of electrical and computer engineering at the North Dakota State University, Fargo, North Dakota. His research interests include optimization, robustness, and security of: cloud, grid, cluster and big data computing, social networks, wired and wireless networks, power systems, smart grids, and optical networks. His work has appeared in more than 250 publications. He is a fellow of the Institution of Engineering and Technology (IET, formally IEE), a fellow of the British Computer Society (BCS) and a senior member of the IEEE.

Jinjun Chen received the PhD degree in computer science and software engineering from Swinburne University of Technology, Australia. He is an associate professor in the Faculty of Engineering and IT, University of Technology Sydney (UTS), Australia. He is the director of Lab of Cloud Computing and Distributed Systems. His research interests include cloud computing, big data, workflow management, privacy and security, and related various research topics. He is a senior member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.