INSTITUT FÜR INFORMATIK

CDOXplorer: Simulation-Based Genetic Optimization of Software Deployment and Reconfiguration in the Cloud

Sören Frey, Florian Fittkau, and Wilhelm Hasselbring

Bericht Nr. TR_1807
September 2018
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

CDOXplorer: Simulation-Based Genetic Optimization of Software Deployment and Reconfiguration in the Cloud

Sören Frey, Florian Fittkau, and Wilhelm Hasselbring

Bericht Nr. TR_1807
September 2018
ISSN 2192-6247

e-mail: hasselbring@email.uni-kiel.de

CDOXplorer: Simulation-Based Genetic Optimization of Software Deployment and Reconfiguration in the Cloud

Sören Frey, Florian Fittkau, and Wilhelm Hasselbring

***Abstract*—Migrating existing enterprise software to cloud platforms involves the comparison of various cloud deployment options (CDOs). A CDO comprises a combination of a specific cloud environment, deployment architecture, and runtime reconfiguration rules for dynamic resource scaling. Our simulator CDOSim can evaluate CDOs, e.g., regarding response times and costs. However, the design space to be searched for well-suited solutions is very large. In this paper, we approach this optimization problem with the novel genetic algorithm CDOXplorer. It uses techniques of the search-based software engineering field and simulations with CDOSim to assess the fitness of CDOs. An experimental evaluation that employs, among others, the cloud environments Amazon EC2 and Microsoft Windows Azure, shows that CDOXplorer can find solutions that surpass those of other state-of-the-art techniques by up to 60%. Our experiment code and data and an implementation of CDOXplorer are available as open source software.**

***Index Terms*—Cloud computing, search-based software engineering, deployment optimization.**

I. Introduction

THE disruptive cloud computing paradigm paves the way for approaching the long-desired idea of utility computing [1]. In the last years, it gained considerable attention from industry and in academia. Along with the steadily increasing interest in cloud computing there also emerged an enormous demand for leveraging cloud technologies for existing (legacy) systems [2]–[4] toward microservice architectures [5]–[8].

S. Frey is with Daimler TSS GmbH, Wilhelm-Runge-Straße 11, 89081 Ulm, Germany. E-mail: freys@acm.org

F. Fittkau is with PPI AG, Wall 55, 24103 Kiel, Germany. E-mail: florian.fittkau@gmx.de

W. Hasselbring is with the Software Engineering Group, Department of Computer Science, Kiel University, Christian-Albrechts-Platz 4, 24118 Kiel, Germany. E-mail: hasselbring@email.uni-kiel.de

A. Problem Description

Migrating and deploying enterprise software to the cloud still entails a wealth of challenges and potential pitfalls. For example, it is tedious to select an adequate cloud environment and the best-suited virtual machine (VM) instance types—with regard to inevitable trade-offs between costs and performance—from the plethora of available cloud offerings [9].

Furthermore, the application and deployment architecture have to be reworked to conform with the chosen cloud and to enable compliance with defined service level agreements (SLAs) and the included quality of service (QoS) stipulations [10]. To exploit the cloud's elasticity and the usually employed pay-per-use model, it is necessary to implement and calibrate *reconfiguration rules* for cost-efficient dynamic resource scaling according to observed usage patterns [11]. For example, a simple reconfiguration rule can be formulated as follows: "*Start R_1 new VM instances of VM instance type R_2 if the average CPU utilization $\geq R_3$% for R_4 minutes.*" This exemplary reconfiguration rule includes the four degrees of freedom R_1-R_4. One of the many possible configurations could allocate $R_1 = 2$, $R_2 = m1.medium$, $R_3 = 80$, and $R_4 = 5$, for instance.

All of those design decisions, such as the selection of a specific cloud environment, component mapping, and a set of reconfiguration rules, are subsumed in so-called *cloud deployment options (CDOs)*. The multitude of potential CDOs need to be explored for well-suited candidates. Unfortunately, techniques for automatically evaluating all CDOs do not exist and a comprehensive manual analysis is most often inapt due to time and budget constraints [12]. Furthermore, deployment optimization problems that consider the QoS-aware composition of an application's services are intractable as they are known to be NP-hard [13].

The design space that spans for all possible CDOs is huge and the elements of a single CDO exhibit complex non-linear interdependencies. Discrete event

simulation is a powerful tool that can help to reveal a CDO's characteristics. However, CDO simulation runs are time-consuming and can take from a few minutes to several hours. Hence, simulating a great number of CDOs is most often still not a viable option and it is therefore very likely that a suboptimal solution is chosen. Moreover, there usually exists no single CDO that causes, e.g., the lowest costs along with the lowest average response times and the lowest number of SLA violations. Thus, a potential cloud user is interested in automatically finding the most adequate trade-off solutions among which the CDO candidate can be selected that best suites the user's specific needs. The set of these most adequate trade-off solutions constitutes a pareto optimum. The included CDOs cannot be improved concerning one objective without deteriorating another objective, e.g., considering a trade-off between costs and response times.

Thus, search-based software engineering seems to be appropriate for cloud engineering [14]. However, existing approaches do not leverage cloud simulators together with reconfiguration rule optimization.

B. Approach and Contribution

In this article, we present the genetic algorithm *CDOXplorer* that explores the CDO search space on the basis of automatically extracted architectural models and approximates the corresponding pareto optimum. Similar problems are addressed by methods of the search-based software engineering field, where genetic algorithms are widely used [15].

In our previous work, we introduced the simulation tool CDOSim [16] that implements a phase of our cloud migration approach CloudMIG [17]–[20]. CDOSim facilitates the simulation of CDOs for determining their respective response times, costs, and SLA violations. We integrated CDOSim into our tool CloudMIG Xpress that provides support for CloudMIG. With CloudMIG Xpress, CDOs can be simulated on the basis of a reverse-engineered architectural system model with monitored or synthetic workload, but CDOs so far had to be configured manually.

CDOXplorer, presented in this article, is implemented in our tool CloudMIG Xpress and supports IaaS-based cloud environments [21], where the most often used building blocks are VMs. In general, genetic algorithms group the candidates—so-called individuals—in populations and use a fitness function to assess the candidates. Then, the best-suited individuals are selected. They reproduce through so-called mutation and crossover operations and after several generations, the individuals that inherited superior properties become dominant. To assess the fitness of CDOs, CDOXplorer uses simulation runs of CDOSim to restrict the search space and to steer the exploration towards promising CDOs. Thus, CDOSim is no longer used only for analyses, but for design exploration purposes as well. By incorporating CDOSim, CDOXplorer is a member of the simulation-based optimization class [22].

Here, the evaluation of the used fitness function is, in contrast to many genetic algorithms, very expensive and requires strict limitations regarding the population size and number of included generations. CDOXplorer not only optimizes the allocation of software components to VMs, but also searches for reconfiguration rules that are aligned with the cloud's elasticity and the specific performance and pricing models of the available cloud environments. A common challenge in the design of genetic algorithms becomes apparent as they do not guarantee to converge to a global optimum, especially, if a low number of generations is used. Hence, we experimentally evaluate the applicability and convergence properties in comparison to other well-known search and optimization algorithms. We report on case studies that employ an open source enterprise resource planning (ERP) system to be deployed on the cloud environments Amazon EC2, an Eucalyptus cluster as private cloud, and Microsoft Windows Azure.

The present article is a substantially extended version of a previously published overview of CDOXplorer [23]. It builds on the authors' further previous work [24] and augments important details of the algorithm's structure, functioning, and quality characteristics. In summary, our main contributions are:

- A simulation-based genetic algorithm (CDOXplorer) for finding near-optimal cloud deployment architectures and runtime reconfiguration rules for enterprise software
- An implementation of CDOXplorer within the scope of our open source tool CloudMIG Xpress, that utilizes models which can, to a great extent, be extracted automatically
- Extensive experiments that employ well-known clouds show that CDOXplorer can find results that surpass those of other state-of-the-art techniques by up to 60%
- An analysis of the search space to be explored by CDOXplorer
- A detailed description of the genetic algorithm's hybridity and adaptivity characteristics that allow

for an online, automatic self-optimization regarding fundamental control parameters
- An in-depth explanation of CDOXplorer's central crossover and mutation operators

The last three items listed above form additional contributions in comparison to the previously published overview of CDOXplorer [23]. CloudMIG Xpress together with our experiment code and data are available online as open source software such that interested researchers may repeat or extend our experiments.[1]

C. Organization

The paper is organized as follows. We provide a motivating example in Section II. The input models and output models for our genetic algorithm are described in Section III. Section IV details CDOXplorer. The experimental evaluation of CDOXplorer is described in Section V. Section VI discusses related work, before our conclusions are drawn in Section VII.

II. Motivating Example

We discuss the example deployment of a software system that is shown on the left side of Fig. 1. This system should be migrated to the cloud environment Amazon EC2. It consists of eight software components that are currently deployed to three interconnected on-premise server machines. Those machines are also called *status quo nodes* as they constitute elements of the deployment architecture that describes the status quo assignment of components to physical machines. We regard all of the components that are deployed to a single status quo node as a separate *service*. A service is an atomic unit concerning the allocation to a VM. This design decision was made to (1) prevent a further explosion in the number of combinations and CDOs that have to be searched, and to (2) render pervasive changes unnecessary that may be required when distributing tightly connected components over different VMs. When migrating such a service to the cloud, it can be deployed on one or more VMs and can be used by zero or more other services, for example, through integrating additional communication mechanisms. Furthermore, we assume that the system fulfills some basic preconditions, for example, that it can run on one of the operating systems that are supported by Amazon EC2. As it complies with those preconditions in our example, the system is *cloud compatible* [18] concerning Amazon EC2 and can be deployed to some of its VMs.

Thus, the three status quo nodes and the deployed components shown in the left part of Fig. 1 result in the three services *Service 0* to *Service 2* in the right part of Fig. 1 that exhibit a similar assignment of components. As can be seen in the lower right part of Fig. 1, a single VM can host multiple services. In our example, it was decided to consolidate *Service 1* and *Service 2* into a joint VM that is started with Amazon EC2's VM instance type *m1.large*. Such VM instance types describe the hardware resources that are available to VMs. The *Service 0* in Fig. 1 is deployed to an own VM that builds upon the *m1.medium* VM instance type.

The basic CDO of Fig. 1 is now given by the number of chosen VM instances, the assignment of services to these VM instances, and the selection of a VM instance type for each VM instance. For this small motivating example, runtime reconfiguration rules are omitted for the sake of simplicity.

Reasoning about the broader array of all potential CDOs for, e.g., just the single cloud environment Amazon EC2, 12 of its VM instance types, and no reconfiguration rules, reveals the general complexity of CDO analysis. When assuming up to three VM images that contain combinations of the three services and that up to two VMs can be started from a VM image, these restrictive settings already yield 4,741,632 CDO candidates (cf. Section IV-F). Without using appropriate heuristics, all CDOs would have to be simulated for reliably finding competitive solutions. For example, assuming a simulation run takes only one minute, simulating these CDOs sequentially would last approximately for nine years.

III. Model Description

CDOXplorer takes a reverse engineered architectural model, a status quo deployment model, a workload profile, and a set of cloud profiles as input and produces a pareto-optimal set of CDOs as output. The input models have to be provided to CDOXplorer such that it can find well-suited CDOs, i.e., CDOXplorer computes a set of near-optimal CDOs that approximate the true pareto-optimal set of CDOs (right side of Fig. 2, see Section III-B). The four types of input models used by CDOXplorer are illustrated on the left side of Fig. 2 and explained in Section III-A, while the output models are illustrated on the right side of Fig. 2 and explained in Section III-B. Central assumptions that underlie the creation and optimization of CDOs

[1] http://www.cloudmig.org

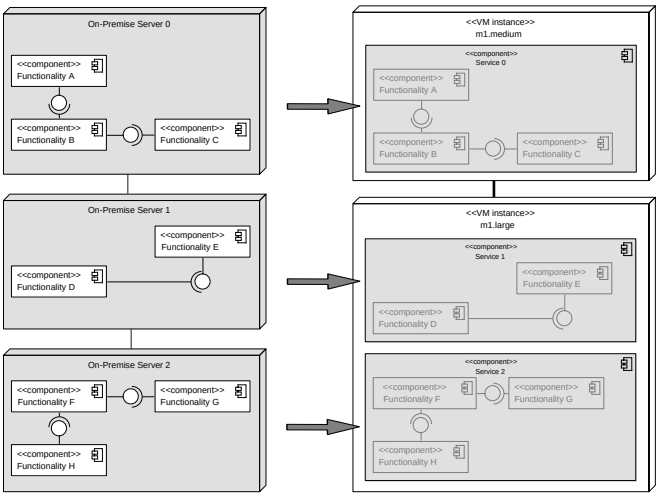


Fig. 1. Mapping on-premise servers and deployed components (left) to atomic services in a basic CDO example (right)

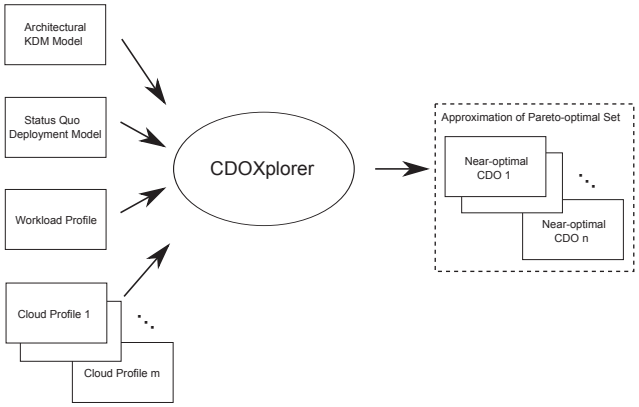


Fig. 2. Input (left side) and output (right side) models of CDOXplorer

with CDOXplorer and that provide the context for actually implementing CDOs for particular software systems and cloud environments are described in Section III-C.

A. Input Models

The input models comprise an architectural model (based on the Knowledge Discovery Meta-Model, KDM [25]), a status quo deployment model, a workload profile, and a set of cloud profiles. We provide tool support for creating these input models, they are described in the following.

An *architectural KDM model* can be automatically extracted by CloudMIG Xpress from an existing system's source code. Currently, we support Java, C#, and Python by incorporating, among others, the reverse-engineering tool MoDisco [26]. CloudMIG Xpress generates architectural models that conform to the ISO/IEC 19506 standard Knowledge Discovery Meta-Model (KDM) that was developed by the Object Management Group (OMG). With KDM, several aspects of software systems can be modeled in a language-independent way, for example, the runtime platforms and source code elements.

To describe the current deployment, a *status quo deployment model* is designed with the integrated editor. Here, KDM code elements can be assigned to status quo nodes. To measure and specify the performance capabilities of these computing nodes, we introduced a benchmark that measures the so-called *mega integer plus instructions per second* (MIPIPS) [16]. The benchmark has to be executed on the on-premise server machines and the results have to be specified in the status quo deployment model by a user. The computing power must be specified to enable CDOSim to interpret a *workload profile* that describes service calls and response times and can be imported from monitoring log files with historical usage data to support the migration [27]. Currently, we support Kieker [28] log files, but additional monitoring log formats can easily be incorporated via plug-ins. If no real monitoring data is present, CloudMIG Xpress also allows for the definition of synthetic workload profiles. For specifying cloud environments, so-called *cloud profiles* are used that, e.g., describe a VM instance types' availability in different geographical areas, specific costs, and performance capabilities. MIPIPS can also be measured by executing the mentioned benchmark for each VM instance type of a cloud environment.

B. Output Models

This section explains the models that are included in CDOs as delivered by CDOXplorer. The models that constitute a CDO are described in Section III-B1. Regarding dynamic resource scaling, a CDO determines a specific scaling type, as detailed in Section III-B2.

1) CDO Models: The basic meta model of CDOs is shown in Fig. 3. A *Cloud Deployment Option* refers to a single *Cloud Environment* and contains so-called *Node Configuration*s. A *Node Configuration* describes specifics of a VM instance, e.g., an included *Service Composition* container refers to the *Service*s that are deployed on this VM. To link *Service*s with the represented source code, they reference parts of the extracted KDM elements.

Furthermore, an *Initial Start Config* specifies the VM instance type that should be used for a VM and also the number of VM instances that have to be

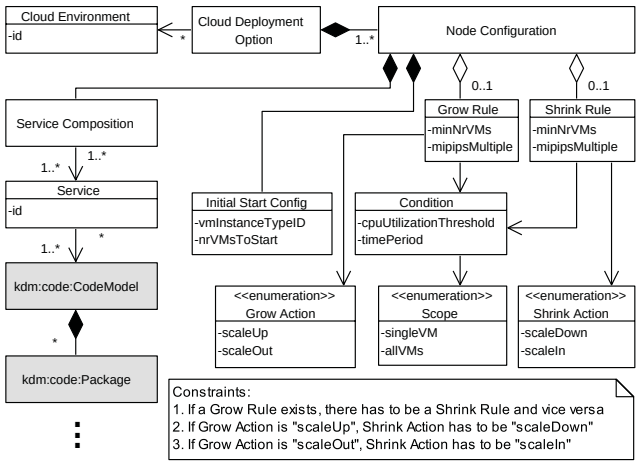


Fig. 3. Basic meta model of CDOs

started initially with this configuration. Moreover, a *Node Configuration* may contain a *Grow Rule* together with a *Shrink Rule*. They represent basic parts of a reconfiguration rule and, from a high-level view, determine how and when computing power is added (*Grow Rule*) or removed (*Shrink Rule*). These rules also specify a minimum number of VM instance types that have to be present and refer to the elements *Grow Action* and *Shrink Action* defining the reconfiguration actions that have to be used for scaling [29], [30]. The reconfiguration actions have to comply with the defined scaling types. Hence, a *scale-up* action can only be used in combination with a *scale-down* action. The same applies to *scale-out* and *scale-in* actions. The different scaling types and the interplay with *Grow Rule*'s and *Shrink Rule*'s *mipipsMultiple* attribute is detailed in Section III-B2.

The actual execution of a reconfiguration action is triggered by *Condition*s (see Fig. 3) becoming *true*. These *Conditions* are defined with the help of a CPU utilization threshold and a time period. For example, concerning a scale-out *Grow Action*, an additional VM instance should be started when the CPU utilization lies above 80% for at least 20 minutes. In this context, the *Scope* element (see Fig. 3) would define whether the 80% refer to the specific VM or to the average of all VM instances that were started from the corresponding *Node Configuration*.

The concrete CDO example in Fig. 4 shows an extended version of the CDO from Fig. 1. A reconfiguration rule that uses vertical scaling (scaling up and down, see Section III-B2) was added to the VM that contains the *Service 1* and *2*, but not to the other VM that hosts *Service 0*. The example reconfiguration grow rule starts a new VM instance when the CPU utilization stays above 80% for at least 20 minutes.

2) Scaling Types: The CDOs produced by CDOXplorer specify a cloud deployment model and a set of reconfiguration rule models. In IaaS-based cloud environments (Infrastructure as a Service), resources can be dynamically acquired and released by executing *reconfiguration actions* to counteract under- and overprovisioning, to benefit from the pay-per-use model, and to ensure the compliance with specified SLAs. Two reconfiguration actions together define one of the scaling types *horizontal scaling* or *vertical scaling* that are shown in the examples of Fig. 5a and 5b, respectively. These scaling types are described in the following:

Horizontal Scaling employs VMs that use the same VM instance type. VM instances are added (*scale-out*) or shut down (*scale-in*). Fig. 5a shows an example where VMs are used that all correspond to the *m1.large* VM instance type.

Vertical Scaling: In contrast to horizontal scaling, the reconfiguration actions *scale-up* and *scale-down* are available for vertical scaling (Fig. 5b). Scaling up adds more resources to a VM, such as more CPU power or more memory, whereas scaling down removes resources. However, to the best of our knowledge, almost no cloud environment currently provides support for arbitrarily exchanging the VM instance type of a VM during operation. Though, the described semantics can be emulated through starting a new VM from a VM instance type whose MIPIPS value is higher (scale-up) or lower (scale-down) than that of the old VM instance. When the new VM instance finishes the startup procedure, the previously used VM is shut down. In the example of Fig. 5b, the MIPIPS value of *m2.4xlarge* is higher than that of *m1.large*.

When applying vertical scaling, the *mipipsMultiple* attribute of *Grow Rule*s and *Shrink Rule*s (see Fig. 3) becomes relevant for defining the VM instance type of the new VM that has to be started. This VM instance type is obtained by multiplying the MIPIPS value of the current VM instance's type with *mipipsMultiple* and rounding the result to the nearest MIPIPS value of the intended VM instance type. The VM instance that is used as a reference point at a particular point in time is called the *Reference Point VM Instance (RPVMI)*.

An example that demonstrates the effect of using different *mipipsMultiple* values for scaling up RPVMIs is shown in Fig. 6. Starting from a VM instance of type *m1.small* that, in this example, has 50 MIPIPS, the subsequently selected VM instance types differ according to the used *mipipsMultiple* value when scaling up. Setting the *mipipsMultiple* value to 1.6, VM instances of the types *m1.medium*, *m1.large*,

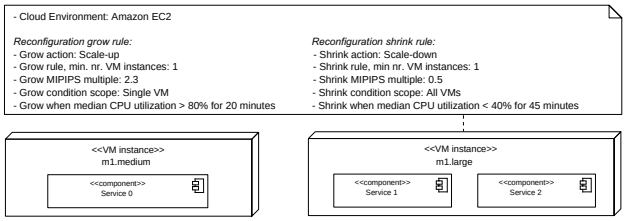
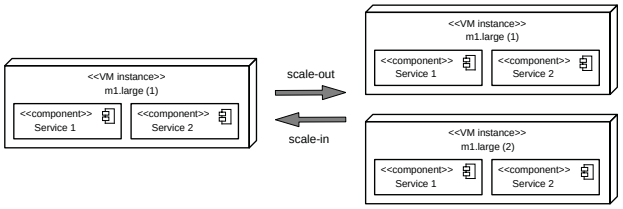
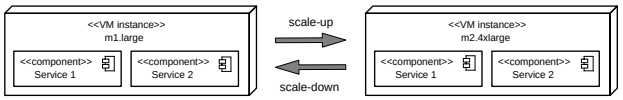


Fig. 4. CDO example of Fig. 1 with added reconfiguration rule



(a) Horizontal scaling: reconfiguration actions scale-out and scale-in



(b) Vertical scaling: reconfiguration actions scale-up and scale-down

Fig. 5. Examples of the different scaling types

and *m2.4xlarge* are started in this order when executing corresponding scale-up reconfiguration actions. For example, the *m1.medium* VM instance type has 75 MIPIPS. Setting a corresponding VM instance as the RPVMI and scaling up the VM instance with *mipipsMultiple* set to 1.6 results in a request to start a VM instance whose MIPIPS value is nearest to $120 = 75 \cdot 1.6$. As the *m1.large* VM instance type exhibits the nearest MIPIPS value (100), a VM instance of the corresponding type is used for scaling up. In contrast, starting from the VM instance of the VM instance type *m1.small* using a *mipipsMultiple* value of 2.0 results in starting a VM instance of the type *m1.large* and then of the type *m2.4xlarge*. Hence, a VM instance of the *m1.medium* type is bypassed.

It should be noted that, despite of the differences between vertical and horizontal scaling, the service composition is retained on a new VM instance for both of the scaling types.

C. Underlying Assumptions

This section describes four fundamental assumptions (AS1-AS4) that underlie the creation and optimization of CDOs with CDOXplorer. As CDOXplorer produces models that describe well-suited CDOs (see Section III-B), those assumptions

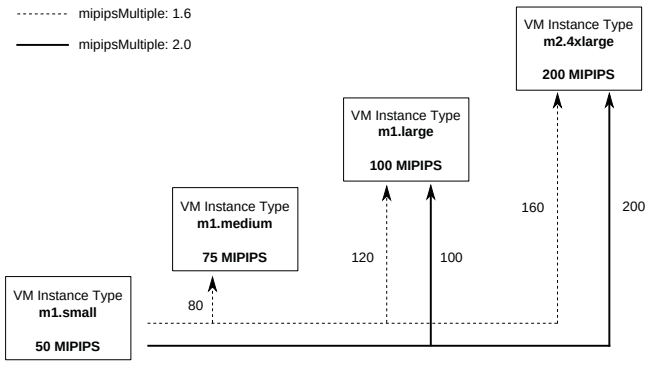


Fig. 6. Effect of using different *mipipsMultiple* values for scaling up. The values beside the arrows indicate the MIPIPS values that result when multiplying a given *mipipsMultiple* with the MIPIPS value of a particular RPVMI's VM instance type

provide the context for actually implementing those CDOs for particular software systems and cloud environments.

AS1: It is assumed that specific recurring usage patterns are known or can at least be estimated while planning a migration to the cloud, as the reconfiguration rules—that are executed at runtime when a system is actually migrated—are aligned to these usage patterns. Users considering a migration to the cloud are presumably interested in two predictive values regarding a specific cloud environment. The first demanded value indicates how the application would perform (and to what costs), if it would be deployed to the cloud environment widely unmodified. Moreover, they are most likely interested in the possible improvement that could be achieved by implementing the best-suited solution. Thus, reasoning about an optimal deployment and reconfiguration rules during the planning phase of a migration is a worthwhile endeavor. In summary, for planning a migration, it is assumed that the CDO optimization is performed offline with the help of known usage patterns, whereas the produced reconfiguration rules are supposed to be executed online after the actual migration (during operation) to adapt to varying workload.

Additionally, CDOXplorer could also be used for continuously modifying the reconfiguration rules at runtime in a self-adaptive fashion, for example, based on predictions of a common workload forecasting methodology, such as an Autoregressive Integrated Moving Average (ARIMA) model or exponential smoothing [31]. In this case, CDOXplorer would be integrated in a way that continuously reruns its optimization procedure for modifying the reconfiguration

rules at runtime. However, such a usage scenario of CDOXplorer, while still being possible, is not in the primary focus of this paper.

AS2: It is assumed that the existing enterprise software system can be adapted towards a scalable architecture with reasonable effort. Alternatively, substantial reworking or the deployment of more coarse grained services may be necessary for monolithic systems.

AS3: It is assumed that a load balancer is available that keeps track of existing VMs and distributes the workload across these VMs. That means, additional components may have to be created and additional cloud services might have to be integrated for managing the pool of used VMs. A corresponding auxiliary architecture might follow the logical structure that is illustrated in Fig. 7. A so-called *rule engine controller* distributes the reconfiguration rules that were created by CDOXplorer to load balancers. Besides distributing the workload among the VMs, the load balancers are therefore also responsible for starting and stopping VMs because of triggered scaling events.

However, the rule engine controller may also be implemented in a way so it can start and stop VM instances on its own. A scaling event is initiated if a reconfiguration rule's defined condition becomes *true*.

AS4: It is assumed that the defined reconfiguration rules can be implemented with the APIs of any IaaS-based cloud environment. Besides using a cloud environment's native APIs, it is also possible to use cloud APIs such as Deltacloud[2] or JClouds,[3] or supporting techniques such as programming directives for elastic computing [32] that are available for various programming languages.

These are reasonable assumptions if a user intends to migrate an existing enterprise software system to the cloud.

IV. The Simulation-Based Genetic Algorithm

This section details the simulation-based genetic algorithm CDOXplorer that processes the models of the previous Section III. First, Section IV-A describes the algorithm design, before Section IV-B formalizes the optimization problem that is tackled by CDOXplorer. Sections IV-C and IV-D detail the crossover and mutation operator of CDOXplorer, respectively. Its adaptivity and hybridity characteristics are explained in Section IV-E. Finally, a search space analysis is given in Section IV-F.

[2]http://deltacloud.apache.org/

[3]http://www.jclouds.org/

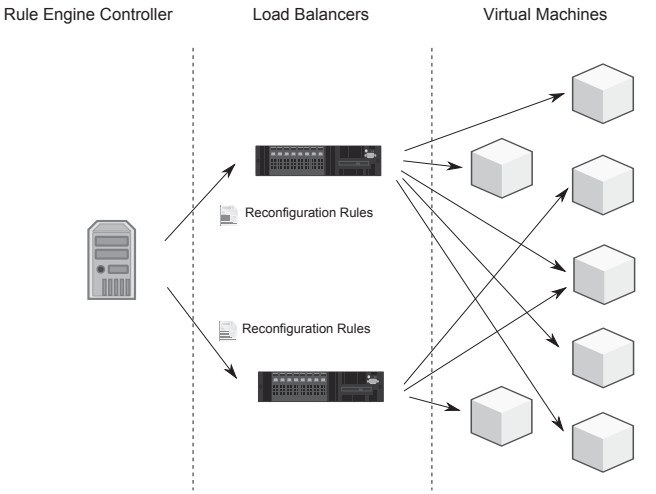


Fig. 7. Logical structure for dynamic resource scaling with reconfiguration rules

A. Algorithm Design

The central purpose of the genetic algorithm CDOXplorer is to efficiently find well-suited CDOs concerning a set of arbitrary IaaS-based cloud environments. In general, genetic algorithms mimic evolutionary processes that describe the advancement of populations over several generations. Evolutionary concepts, such as the survival of the fittest and inheritance of properties that turn out to be advantageous, are included and utilized as optimization techniques. Genetic algorithms are usually used in the context of multi-objective optimization problems. With CDOXplorer, we consider the objectives response times, costs, and SLA violations of CDOs.

As there usually exists no single global optimum, genetic algorithms aim to iteratively approximate the *pareto-optimal set* (also referred to as *pareto optimum*). This pareto-optimal set is a subset of all individuals that includes all *pareto-optimal* individuals, i.e., individuals for which the improvement of one objective (e.g., lower costs) would inevitably lead to a deterioration of another objective (e.g., higher response times). As a consequence, the individuals contained in a pareto optimum constitute trade-off solutions that have to be inspected manually. In general, individuals are compared with so-called fitness functions. After many generations, the fittest individuals become dominant. The reproduction of each

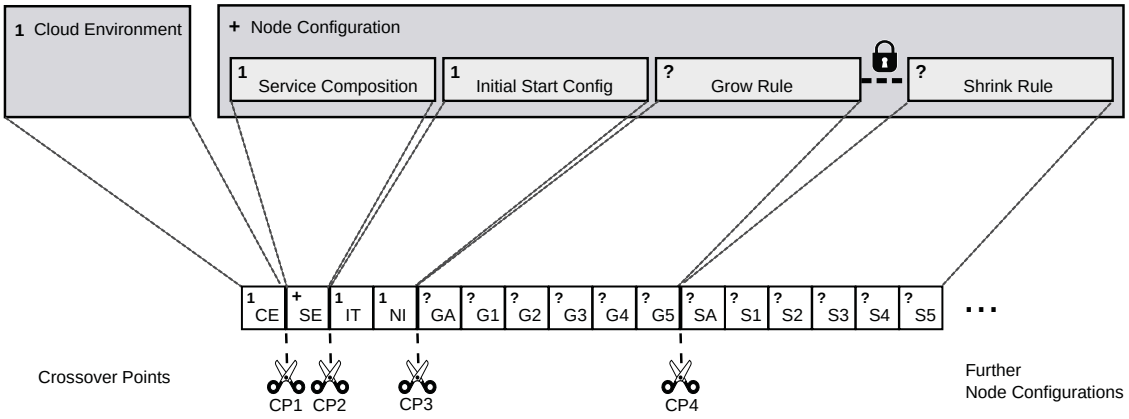


Fig. 8. Compound chromosome overview. Gray boxes: chromosomes, white boxes: genes (listed in Table I). `1`, `+`, `?` in the boxes' upper left corner indicate that the elements occur exactly once, at least once, and at most once, respectively.

generation includes the following four basic steps S1-S4 [15]:

S1 Select parents
S2 Recombine parents (crossover)
S3 Mutate offspring
S4 Evaluate offspring's fitness

The first step S1 selects individuals for reproduction. For CDOXplorer, S1 is based on the selection operation of the NSGA-II algorithm [33] for selecting appropriate pairs of parents and ensuring the diversity of solutions. We apply two tournament rounds for choosing among candidates. Simply put, an individual has to be fitter than at least two others. CDOXplorer produces two children from two parents via executing a custom crossover (S2) and mutation operator (S3) that are detailed in the Sections IV-C and IV-D, respectively.

As mentioned before, CDOSim is used for evaluating the individuals in the fourth step S4. In this step, the values of the objectives that have to be optimized are obtained by simulating CDOs. Because the simulations are very time-consuming, ranging from minutes to hours, we limited the population size and configured the basic parameters of CDOXplorer as follows. Our populations contain 50 individuals (population size α). 25 individuals are selected from each generation for reproduction (number of parents μ) and each generation spawns 50 children (number of children λ). Furthermore, CDOXplorer produces 60 generations per default. Genetic algorithms also use biological analogies for representing the individuals of a population. Their basic elements are specified by *genes*. Considering the classes in Fig. 3, the ID of a service represents a single gene, for instance. All genes together constitute the so-called *genome* that contains the complete genetic information of all possible CDOs. Genes can be grouped in larger structures that are called *chromosomes*. Fig. 8 illustrates the basic chromosomes and genes that are processed by CDOXplorer. The chromosomes correspond to the class structure of Fig. 3 and map to one or more genes that together form a gene sequence. Such a single gene sequence encodes a specific CDO and is called a *genotype*.

The *Node Configuration* chromosome constitutes a container for further chromosomes that correspond to classes shown in Fig. 3. As there can exist one or more node configurations each having zero or one pair of a *Grow Rule* and *Shrink Rule* chromosome, the genotypes exhibit variable lengths. The *crossover point*s in Fig. 8 are detailed in Section IV-C. The abbreviations and range of values that are used for the single genes are listed in Table I. These genes correspond to the attributes of classes from Fig. 3. We limited their values to narrow ranges for avoiding a further growth of the search space.

Fig. 9 shows three examples of CDOs that are encoded as genotypes. Here, the third example `CDO3` corresponds to the CDO that is depicted in Fig. 4. Thus, it contains two node configurations from which only the second exhibits assigned grow and shrink rules, as can be seen by taking into account the general structure of genotypes in Fig. 8. Hence, the cloud environment Amazon EC2 is encoded by the number 7 in this example (gene `CE`) and the first node configuration comprises only the genes 2-4. The deployed Service 0 can be identified by the first gene `SE`. Furthermore, the second node configuration that includes a reconfiguration rule is represented by the genes 5 (`SE`) - 20 (`S5`). The other example `CDO1` shows a genotype using the cloud environment Microsoft Windows Azure (encoded by the number 8 in

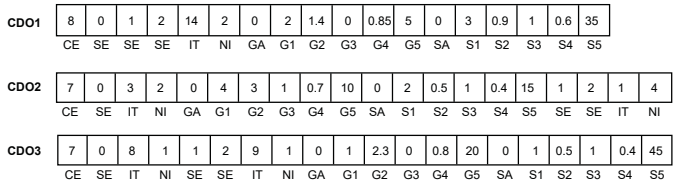


Fig. 9. CDO examples encoded as genotypes

the gene CE) and only one node configuration (genes 2 (SE) - 18 (S5)). The example CDO2 includes two node configurations (first node configuration: genes 2 (SE) - 16 (S5), second node configuration: genes 17 (SE) - 20 (NI)) and uses Amazon EC2 (encoded by the number 7 in the gene (CE)). These genotypes are reused in Sections IV-C and IV-D as examples.

B. Problem Formalization

After introducing the basic design of our genetic algorithm, we can describe the multi-objective optimization problem that is tackled by CDOXplorer as follows. Let Φ be the set of all *feasible* CDOs (feasibility is explained below) for a given set of IaaS-based cloud environments and an architectural model, status quo deployment model, and workload profile of a software system. The goal is to find a CDO $x \in \Phi$ that minimizes the values of the three objective functions $costs(x)$, $rt(x)$, and $sla(x)$ that determine the costs, average response time, and number of SLA violations of x, respectively.

The costs refer to the total amount of monetary units owed to a cloud provider because of utilizing provided services. The response times refer to the average response times of the methods that are included in a workload profile. Lastly, the SLA violations indicate the number of method calls with response times that exceed a given threshold [34]. We denote the set of all node configurations in x as N and the set of all services in x as S. x is feasible if it (1) complies to the structure of CDOs (see Fig. 8), (2) complies to the value ranges defined in Table I, and (3) complies to the constraints that are described below. We will use the following notation where y denotes a gene or chromosome and z denotes a chromosome or CDO. Furthermore, gr names a grow rule and sr a shrink rule.

$y \prec z$: y is contained in z
$\triangle(y, z)$: Number of y in z
T_c: Set of VM instance types of cloud environment c

x has to comply with the following Constraints 1-7.

$$\forall s \in S, x \in \Phi : \triangle(s, x) \geq 1 \quad (1)$$

$$\forall s \in S, n \in N : \triangle(s, n) \leq 1 \quad (2)$$

$$\forall n \in N : \exists s \in S, s \prec n \quad (3)$$

$$\forall IT \prec x, CE \prec x, x \in \Phi : IT \in T_{CE} \quad (4)$$

$$\forall n \in N : \triangle(gr, n) = \triangle(sr, n) \leq 1 \quad (5)$$

$$\forall gr \prec n, sr \prec n, n \in N : gr \succ GA = SA \prec sr \quad (6)$$

$$\forall gr \prec n, sr \prec n, n \in N : gr \succ G4 > S4 \prec sr \quad (7)$$

Constraint 1 describes that each service has to be present at least once in some node configuration of an individual. Furthermore, we do not allow duplicated services in a single node configuration (Constraint 2). The Constraint 3 states that at least one service has to be present in each node configuration. A specific VM instance type (gene IT) also has to conform with a stated cloud environment (gene CE, see Constraint 4). Thus, VM instance types of Amazon EC2 cannot be used in conjunction with Microsoft Windows Azure, for instance. Constraint 5 phrases the following limitation: If a grow rule exists in a node configuration, a shrink rule also has to be present in this node configuration and vice versa. Considering grow rules and shrink rules, the grow actions and shrink actions have to match (Constraint 6, cf. Table I), i.e., a scale-out rule has to be accompanied by a scale-in rule and a scale-up rule has to be associated with a scale-down rule. The CPU utilization thresholds of grow rules and shrink rules indicate trigger points when to start or shutdown a VM instance. Here, the CPU utilization threshold of a grow rule has to exceed the CPU utilization threshold of a shrink rule (Constraint 7).

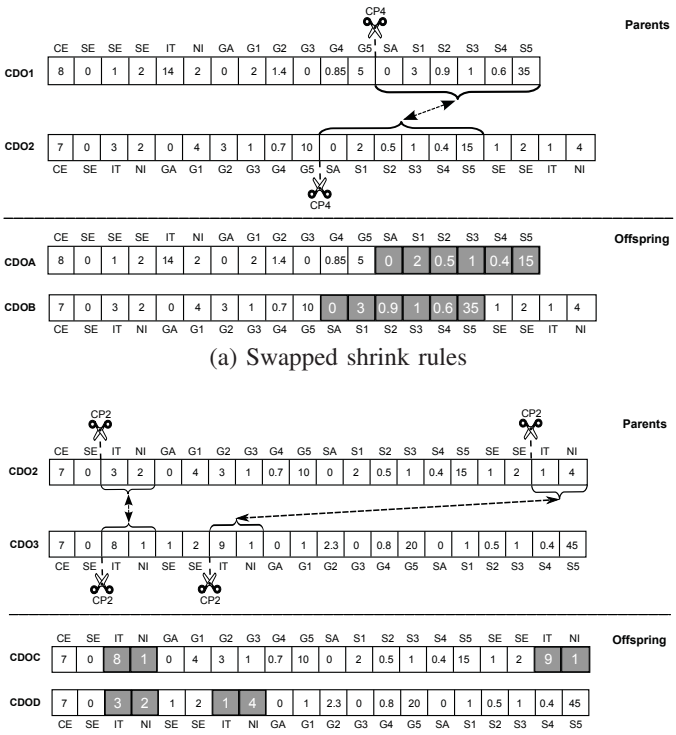
All constraints described before relate to structural properties of CDOs, such as the presence of services in a CDO's node configurations or the exclusion of specific VM instance types. These structural properties can be checked by CDOXplorer after producing new offspring without having to perform costly simulation runs. However, CDOXplorer can also be extended to allow for hard constraints that cover dynamic properties of CDOs while accepting an increased runtime due to additional simulation runs. For example, a possible extension could render a CDO infeasible if it exceeds a given threshold regarding the caused number of SLA violations instead of only assigning an inferior fitness score to that CDO.

C. Crossover Operator

The reproduction procedure involves, in the first place, the application of the crossover operator for producing two children from two parent individuals by mixing their genetic information. This technique follows the biological analogy for passing properties

TABLE I
DESIGN OF THE USED GENES

Gene	Range	Description	Chromosome
CE	$\mathbb{N}$	Cloud environment id	Cloud Env.
SE	$\mathbb{N}$	Service id	Service Comp.
IT	$\mathbb{N}$	VM Instance type id	Initial Start C.
NI	$\mathbb{N}$	Nr. of VM instances to start initially	Initial Start C.
GA	0,1	Grow action; 0: scale-up, 1: scale-out	Grow Rule
G1	$\mathbb{N}$	Minimum nr. of VM instances	Grow Rule
G2	1.1-3.0	MIPIPS multiple in steps of 0.1	Grow Rule
G3	0,1	Condition scope; 0: single VM, 1: all VMs	Grow Rule
G4	0.05-1.0	Condition median utilization in steps of 0.05	Grow Rule
G5	5-60	Condition time period in steps of 5 minutes	Grow Rule
SA	0,1	Shrink action; 0: scale-down, 1: scale-in	Shrink Rule
S1	$\mathbb{N}$	Minimum nr. of VM instances	Shrink Rule
S2	0.1-0.9	MIPIPS multiple in steps of 0.1	Shrink Rule
S3	0,1	Condition scope; 0: single VM, 1: all VMs	Shrink Rule
S4	0.0-0.95	Condition median utilization in steps of 0.05	Shrink Rule
S5	5-60	Condition time period in steps of 5 minutes	Shrink Rule



(a) Swapped shrink rules

(b) Swapped initial start configurations

Fig. 10. Crossover operator examples

of the parents to their offspring. As both reproduction operators are intended to produce only feasible candidates in CDOXplorer, an arbitrary interleaving of genes is not allowed. Therefore, as a first measure, the mixing of genes is restricted to dedicated positions in the genotype that are called *crossover points*. Four crossover points `CP1`-`CP4` are defined that are also shown in Fig. 8. They get selected by chance, are aligned to the boundaries of the chromosomes, and specify corresponding gene sequences that can be swapped.

Fig. 10 shows two examples for applying the crossover operator with the help of the previously introduced CDOs of Fig. 9. In Fig. 10a, `CP4` was selected for mixing `CDO1` and `CDO2`. As `CDO1` includes only one node configuration and the second node configuration of `CDO2` contains no reconfiguration rules, only the first two shrink rules have to be swapped. An example that considers two CDOs where each contains two node configurations is shown in Fig. 10b. As `CP2` is selected, both initial start configurations of `CDO2` and `CDO3` are swapped.

However, not every crossover point can be used with every combination of CDOs. For example, shrink rules can only be swapped when both parents have at least one shrink rule. Furthermore, exchanging the gene `CE` makes only sense if different cloud environments are used. Otherwise, the crossover operation would produce two new CDOs that are genetically identical to their parents. This would (1) not only introduce no new genetic characteristics to the common gene pool, but would also (2) waste resources as the identical CDOs might be simulated several times yielding the same values of the objective functions. The ranges of values described in Table I and the constraints specified in Section IV-B have to be taken into account. Hence, swapping the gene `CE` implicates also the swapping of the VM instance types (gene `IT`) for retaining consistency, for instance.

Instead of ensuring CDO consistency by way of operator construction, another general possibility, that would allow for the arbitrary interleaving of genes and the intermediate construction of infeasible chromosomes, would be to employ a compensating repair mechanism. Nevertheless, the repair scheme would, similar to our approach, eventually also have to guarantee chromosome consistency [35]. In essence, the

effort for direct constraint handling would be moved to a separate repair operator.

D. Mutation Operator

After two parent individuals have initially produced two children with the help of the crossover operator, the mutation operator is applied to each child. In general, genetic algorithms randomly mutate single genes or gene sequences. This imitates sudden leaps and modifications to the global gene pool that occasionally appear during the evolution process. Considering the influence on the overall optimization procedure, the mutation operator fosters retaining the diversity of the individuals and helps to avoid convergence to a local optimum. Just as the crossover operator, the mutation operator is aligned to the chromosome boundaries that are shown in Fig. 8. As a mutation also has to maintain the inner structure of a chromosome, the mutation operator is divided in the five sub operators M-CE, M-NN, M-IS, M-SC, and M-RR that are described in the following.

M-CE Mutates the cloud environment id (gene CE), i.e., a different cloud environment is used. Here, the IT gene of each node configuration has to be modified as well, as the formerly used VM instance types are not available for the new cloud environment.

M-NN Mutates the number of node configurations and relocates the services. When a node configuration is added, a service (gene SE) is moved from another node configuration to the new one. When a node configuration is removed, all services are relocated to other node configurations.

M-IS Mutates the initial start configuration of a single node configuration, i.e., another VM instance type (gene IT) is selected or the number of VM instances that are initially started with regard to this node configuration (gene NI) is increased or decreased.

M-SC Mutates the service composition of a single node configuration. A service (gene SE) can be added or removed.

M-RR Mutates a reconfiguration rule, i.e., at least one of the genes GA, G1-G5, SA, S1-S5 is modified. When altering a grow rule, changes may also be necessary for the shrink rule to satisfy the constraints (cf. Section IV-B) and vice versa.

Fig. 11 shows two examples that utilize the mutation operator and CDOs from Fig. 9. In Fig. 11a, the sub operator `M-RR` is used in conjunction with `CDO1`. In this example, the median utilization threshold of the grow rule (gene G4) is lowered by 5 %. The example in Fig. 11b applies the mutation sub operator `M-NN` to `CDO3` from Fig. 9 that contains two node configurations. In this example, `M-NN` removes the first node configuration. As the Service 0 is deployed only there, it has to be relocated to the second node configuration to satisfy the Constraint 1.

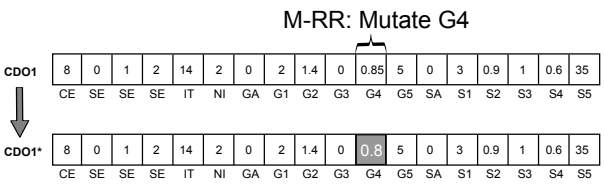
E. Adaptivity and Hybridity Characteristics

CDOXplorer uses adaptive mutation and crossover rates to increase the convergence speed of the genetic algorithm. Adaptive genetic algorithms have already been investigated for a long time and the potential to outperform pure variants was demonstrated in numerous applications [36]–[38]. Important goals of adapting reproduction mechanisms over time are to maintain the diversity of the populations and to retain convergence capabilities. CDOXplorer uses *parameter control* to implement the adaptivity. Parameter control adapts important parameters of a genetic algorithm, such as the mutation and crossover rate, during algorithm execution. In contrast, *parameter tuning* derives static parameters from several precedent test runs.

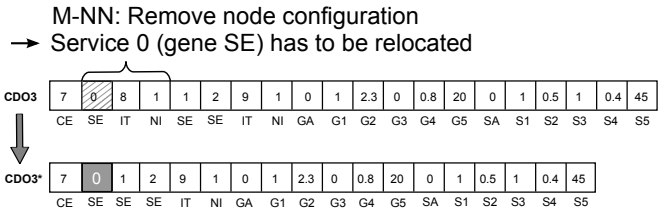
CDOXplorer also employs a combination with a local search technique to further improve its search results. Hence, it is also a hybrid genetic algorithm [39]. Hybrid genetic algorithms incorporate other optimization techniques such as further evolutionary heuristics or gradient-based search. CDOXplorer combines the population-based search of the genetic algorithm with a local search. Similarly to the integration of adaptive mechanisms, combining genetic algorithms with other search techniques has been shown to be beneficial in a large number of application areas, as demonstrated by [40]–[43], for instance. The combination of genetic algorithms with the particular search technique local search is also known to deliver efficient optimization mechanisms [44]–[46].

The rest of this section describes CDOXplorer's adaptivity and hybridity characteristics in Section IV-E1 and Section IV-E2, respectively.

1) Adaptivity: CDOXplorer adapts the reproduction mechanism between subsequent generations to guide the search and to overcome local plateaus of the search space. The reproduction of two parent individuals chosen by the selection procedure is performed only with a certain probability. More specifically, the execution of the crossover and mutation operations are accomplished according to a separate crossover rate (cr) and mutation rate (mr). Instead of using fixed rates, CDOXplorer utilizes dynamically changing crossover and mutation rates. These rates adapt to the evolution of the generations' total fitness. Therefore, CDOXplorer can be classified as adaptive.



(a) The median utilization threshold of a grow rule condition is lowered from 85% to 80%.



(b) The first node configuration is removed during this mutation. As the Service 0 is not present on any other node configuration, it has to be relocated to satisfy the constraints.

Fig. 11. Mutation operator examples

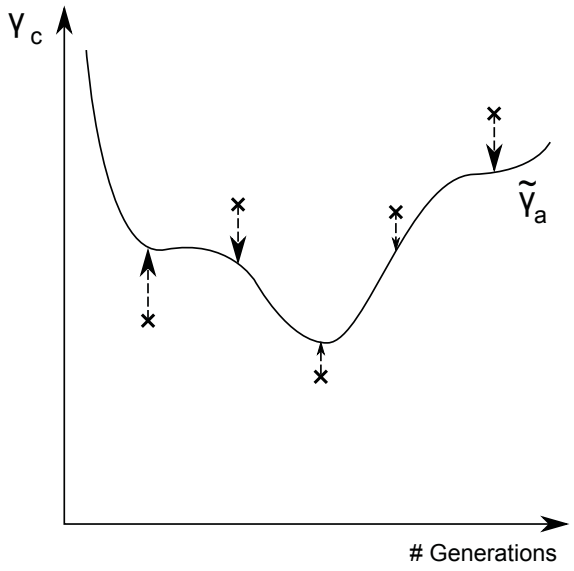


Fig. 12. Tracking the progress of pareto optimum quality improvement over generations to adapt the crossover and mutation rates. Given two subsequent generations g_n and g_{n+1}, regarding g_{n+1}, γ_c denotes the fraction of new elements in g_{n+1}'s pareto optimum regarding g_n. $\tilde{\gamma}_a$ is the median of all γ_c per generation.

Basically, CDOXplorer compares successive generations and determines the fraction of new elements in the new generation's current (temporary) pareto optimum. After the reproduction procedure of a generation finished, the fraction of new elements in the corresponding current pareto optimum in relation to the pareto optimum of the direct predecessor generation is denoted γ_c. In each generation, CDOXplorer removes the 50 individuals with the worst fitness values. Hence, the quality of the pareto optimum becomes better or at least remains stable with every generation and therefore, higher values of γ_c correspond to a higher improvement. Additionally, for each but the first generation, CDOXplorer calculates the median over all previous fractions of new elements (γ_c) that is called $\tilde{\gamma}_a$. The calculation procedures for actually computing the crossover and mutation rates are chosen by comparing γ_c with $\tilde{\gamma}_a$. Two cases C1 and C2 are distinguished, where C1 corresponds to $\gamma_c <= \tilde{\gamma}_a$ and C2 corresponds to $\gamma_c > \tilde{\gamma}_a$. Fig. 12 illustrates an example that shows the progress of the pareto optimal quality improvement by comparing, for each generation, γ_c and $\tilde{\gamma}_a$.

In case *C1)* ($\gamma_c <= \tilde{\gamma}_a$), the new generation only achieved a lower improvement than the other generations (on average) before. cr and mr are therefore set as follows: $cr = 1 - \gamma_c$ and $mr = \gamma_c$. Please note that in general crossover operations have a more disruptive impact than mutation operations. Hence, a lower improvement leads to an increased probability of applying the crossover operator and exploring areas of the search space that are further away ("bigger leaps"). The intention is to investigate essentially new CDO structures that may exhibit fresh potential.

The case C2 is described in the next Section IV-E2, as it represents CDOXplorer's hybridity characteristic.

2) Hybridity: As described in the previous Section IV-E1, CDOXplorer compares γ_c and $\tilde{\gamma}_a$ after the new individuals of each generation got evaluated, i.e., their fitness is determined by simulating the objective values with CDOSim. The comparison of γ_c and $\tilde{\gamma}_a$ distinguishes two cases, C1 and C2.

In case *C2)* ($\gamma_c > \tilde{\gamma}_a$), the current pareto-optimal set changed more than the median of all previous changes. Therefore, the currently explored search space area appears to be promising and is now examined carefully and with a more fine-grained resolution. Thus, CDOXplorer avoids performing the disruptive crossover operation and restricts the mutation operation to `M-RR` (reconfiguration rule, see Section IV-D), that is in turn executed each time ($mr = 1$) with small changes to the reconfiguration rules. The changes to the reconfiguration rules therefore affect the genes `GA`, `G1-G5`, `SA`, and `S1-S5`. Those changes usually have a lower impact on the overall results than, for example, using a different cloud provider or relocating services to new VM images. A modification that shuts down a VM instance after 25 instead of 20 minutes of lower CPU utilization (gene `S5`), for instance, influences the objective values to a lower extent. The modified reconfiguration rules therefore constitute the direct neighborhood of a specific CDO, as they repre-

sent solutions that are located in its direct surrounding of the search space. The thorough and structured investigation of this direct surrounding constitutes a local search. The combination with this meta-heuristic optimization method classifies CDOXplorer as *hybrid*.

F. Search Space Analysis

Table I shows that CDOXplorer employs a discrete instead of a continuous search space. Nevertheless, the size of the search space is remarkable. Because of the expensive simulation-based evaluation function, the scale of the search space is of peculiar interest for analyzing the complexity of the search problem that is tackled by CDOXplorer, as the simulation considerably limits the speed for processing diverse areas.

The surface of the search space is rugged, as already small changes of a gene's value (cf. Table I) may lead to considerably different results. For example, launching new VM instances slightly too late may cause, under adverse conditions, overflows in queues that buffer requests and that cannot be recovered subsequently. Hence, a corresponding small change may occasionally result in a significant increase in SLA violations.

The size of the search space that has to be analyzed by CDOXplorer depends on various factors of specific cloud migration scenarios. For example, considering more cloud environments as potential target candidates for a cloud migration scenario significantly expands the search space. To support reasoning about the size of the search space, the already introduced notion concerning the set of all feasible CDOs Φ and the set of all distinct services S is utilized. Hence, given the cardinality $|S|$ of S and the Constraint 3 (see Section IV-B), there exist $2^{|S|}-1$ different combinations of services that can be deployed on each node configuration. Furthermore, let c be the number of available IaaS-based cloud profiles and v the maximum number of VM instances that are allowed to start initially in a simulation of CDOSim. Regarding a cloud profile i, let a_i be the number of allowed node configurations for i, and let t_i be the number of VM instance types defined in i. Then, the number of all feasible CDOs $|\Phi|$ is given by Equation (8):

$$\begin{aligned}|\Phi| &= \sum_{i=1}^{c}((2^{|s|}-1)\cdot t_i v \cdot (1+2\cdot \\ &\quad \overbrace{3\cdot 20\cdot 2\cdot 12}^{G1-G3;G5}\cdot \overbrace{3\cdot 9\cdot 2\cdot 12}^{S1-S3;S5}\cdot \overbrace{\sum_{j=1}^{20} j}^{G4;S4}))^{a_i} \\ &= \sum_{i=1}^{c}((2^{|s|}-1)\cdot 391{,}910{,}401\cdot t_i v)^{a_i}\end{aligned} \tag{8}$$

Each summand of the outer sum represents the number of feasible CDOs for a specific cloud profile i. $2^{|S|}-1$ different combinations of services can be deployed on t_i different VM instance types and up to v VM instances can be started initially for each of those configurations. Furthermore, a node configuration does not need to have reconfiguration rules. This case is represented by "1" in the inner braces. Otherwise, when the grow and shrink rules are used, either a *scale-out/scale-in* or a *scale-up/scale-down* combination can be applied (two possibilities). CDOXplorer is configured in a way that allows the genes `G1` and `S1` to take values from 1 to 3. Further on, the value ranges in Table I show that `G2` contributes 20 and `S2` 9 possibilities, as well as the number of the possibilities for `G3`, `S3`, `G5`, and `S5`. `G4` and `S4` account for the last inner sum, as additionally to their value ranges, Constraint 7 has to be considered.

The scenario of Fig. 1 was already used in Section II to indicate the size of the search space. Here, we present the calculation of the search space size with the help of Equation (8). The scenario includes a status quo deployment with three status quo nodes that result in the three services *Service 0* to *Service 2*. The following assumptions were made:

- *Cloud environment(s)*: Amazon EC2 ($c=1$)
- *VM instance type(s)*: 12 VM instance types ($t_i=12$) of Amazon EC2 ($i=1$)
- *Reconfiguration rule(s)*: 0 (ignore parts for `GA`, `G1-G5`, `SA`, `S1-S5`) in (8)
- *VM image(s)*: 3 ($a_i=3$)
- *Max. init. VM instance(s)*: 2 ($v=2$)

Hence, the number of possible CDOs for the example scenario described in Section II is given by:

$$|\Phi| = \sum_{i=1}^{1}((2^3-1)\cdot 12\cdot 2\cdot 1)^3 = 4,741,632 \tag{9}$$

V. Experimental Evaluation

To evaluate our simulation-based genetic algorithm CDOXplorer, we implemented it as a component

of our open source tool CloudMIG Xpress. We let CDOXplorer create CDO candidates on the basis of three well-known cloud environments and investigated the following research questions:

RQ1: Feasibility—Does CDOXplorer reliably provide well-suited results?

RQ2: Competitiveness—Are CDOXplorer's results at least on a par with those from other state-of-the-art search methods?

CDOXplorer utilizes our simulator CDOSim [16]. Please note that the following evaluation covers only CDOXplorer. For an evaluation of CDOSim that demonstrates its applicability and precision, please refer to [16], [47]. We begin in Section V-A with explaining the methodology that we used to tackle the research questions. The experimental setting is then described in Section V-B. The research questions RQ1 and RQ2 are detailed in Sections V-C and V-D, respectively, before the threats to validity are discussed in Section V-E.

A. Methodology

We used the Opt4J framework for meta-heuristic optimization [48] as a basis for our implementation of CDOXplorer and also applied the NSGA-II algorithm for the selection of parent individuals (see Section IV-A). As NSGA-II utilizes so-called fronts of non-dominated individuals [33], a pareto optimum is also called *pareto-optimal front* in this context. There exist several standard performance metrics that basically measure the level of approximation towards the pareto-optimal front. As all existing performance metrics exhibit different strengths and weaknesses and therefore are often used in combination [49], we decided against using a single metric. Instead, we applied the two popular metrics *inverted generational distance* (IGD) [50] and *hypervolume indicator* (HV) [51] that are illustrated in Fig. 13. Usually, the metrics specify the approximation towards the true pareto-optimal front (PF_{true}) if this is known. In our case, PF_{true} can only be obtained by simulating all feasible CDOs. As this is not possible in a reasonable amount of time, we instead use the best pareto-optimal front, that is formed by the combination of the overall simulation results, as a proxy, and term it OPF_{Best}. Another pareto-optimal front depicted in Fig. 13 is PF_{Known}. It is the result of each execution of CDOXplorer, whereas each execution includes 3,000 CDOSim simulations. The cube's axes in Fig. 13 indicate the three objectives that are optimized by our algorithm: costs, response times, and SLA violations.

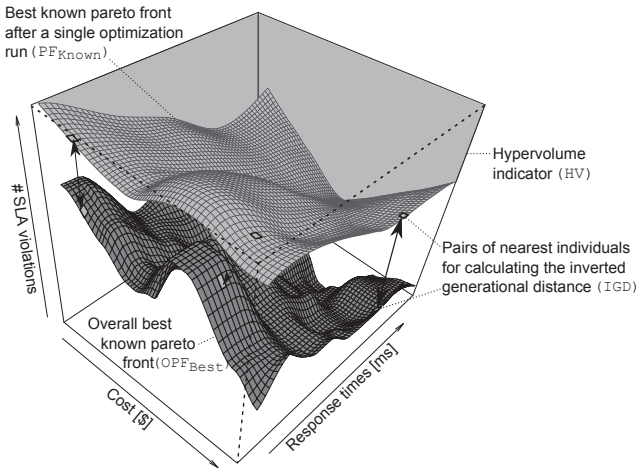


Fig. 13. The metrics *inverted generational distance* and *hypervolume indicator* (light gray colored volume) are used for evaluation.

Starting from the individuals in OPF_{Best}, the nearest individuals in PF_{Known}, in terms of the Euclidean distance, are used for calculating the IGD metric. This metric directly measures the distance from OPF_{Best} to PF_{Known}. Hence, smaller values indicate a better approximation.

In contrast, the values produced by the HV metric become bigger as the approximations become better. HV measures the volume in hyper-dimensional space that is covered by PF_{Known} regarding a reference point $r \in \mathbb{R}^d$ in d-dimensional space. As we consider three objectives and configure HV accordingly, the mentioned volume corresponds to the volume in three-dimensional space that is colored light gray in the illustration of Fig. 13. HV therefore implicitly rates the distance from OPF_{Best} to PF_{Known} as is also done by IGD. Additionally, both metrics assess the spread of PF_{Known}, that constitutes another important quality characteristic. When covering larger parts of the search space, there exist less unexplored areas that could potentially contain better solutions.

To judge the measurement results of these metrics for CDOXplorer, we used several state-of-the-art search and optimization techniques for comparison. However, as the objective values cannot be obtained by solving functions analytically, some popular classes of approaches, such as gradient-based optimization methods, cannot be used. Though, direct search methods are suited for simulation-based optimization [52]. Therefore, we use the simple yet effective stochastic algorithms *simple random sampling (SI-RS)* and *systematic random sampling (SY-RS)* [53] as two out of the three algorithms used for comparison with CDOXplorer. The algorithm SI-RS creates 3,000 CDO individuals by chance and serves

as a baseline algorithm. SY-RS also produces 3,000 individuals by chance, but works in a different way. It assumes that the elements of the feasible design space can be represented as a partially ordered set, such that its elements can be put in sequence according to an ordering relation. As CDOXplorer employs a notion of locality and exploits the neighborhood relation of CDOs (cf. Section IV-E), SY-RS ensures a uniform coverage of the feasible design space. This is especially important due to the necessary restrictions regarding the population size and number of included generations when utilizing simulation-based optimization. Hence, SY-RS explores the feasible design space in a structured way by sequentially traversing the CDO value ranges of all genes. SY-RS iterates over all f feasible CDO candidates and randomly selects the c-th CDO from the $[0, floor(\frac{f}{3{,}000})]$ interval at the beginning of a CDOXplorer run, where $floor(x)$ rounds $x \in \mathbb{R}_+$ down to the next natural number. The next CDO is then given by the $(c + floor(\frac{f}{3{,}000}))$-th candidate and so forth.

The third algorithm used for comparison purposes is, just as a genetic algorithm, a nature-inspired metaheuristic termed *simulated annealing (SI-AN)* [54]. Basically, it mimics the temperature cooling process of materials. Simulated annealing can be used for both single and multi-objective optimization and can deliver excellent solutions in both cases [55], [56]. In our problem context, SI-AN is implemented as a variant for multi-objective optimization. To emulate the temperature cooling process, SI-AN reuses the mutation sub operators introduced in Section IV-D as illustrated in Fig. 14. The temperature is adapted according to five phases. The first phase delivers feasible CDOs with random chromosomes. The phases 2-5 utilize mutation sub operators with specific probabilities to reduce disruptive modifications over time. For example, triggering the start of an additional VM instance due to exceeding a CPU utilization threshold (M-RR) five minutes later, very likely has a lower impact than using a different cloud environment (M-CE). Hence, SI-AN uses the M-CE operator in an earlier phase than M-RR, for instance. SI-AN also enables a smooth transition between the phases and ensures diversity of the CDO search space by fading in and out the usage of mutation sub operators in subsequent phases instead of simply using a single operator. Hence, SI-AN decreases the probability of utilizing an established operator with the same rate it uses to raise the probability a new operator is used.

CDOXplorer uses 60 generations with populations of 50 individuals. Hence, CDOXplorer applies 3,000

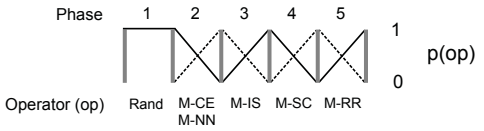


Fig. 14. Simulated annealing reuses mutation sub operators

simulations in a single run. This calibration was guided by the following considerations. As mentioned before, the simulations are very time-consuming, ranging from minutes to hours (in rare cases) for simulating a single CDO. A possible technique to cope with the computational complexity of simulations is to use approximation methods, such as computing surrogate models [57]. However, CDOXplorer targets a highly dynamic context. For example, VM instance types and their capabilities and prices are described in CloudMIG's cloud profiles (see Section III-A). These cloud environment properties change rather frequently.

Hence, pre-computing large numbers of corresponding surrogate models for these cloud profiles is also not feasible. Therefore, reducing the number of generations and population sizes remains as a possible solution [57], while having to ensure the optimization performance, as is done in this experimental evaluation. Several combinations were compared in our intensive preceding lab tests and the combination of 60 generations and 50 individuals provided good results and a viable basis for the huge number of simulation runs that were conducted in the experimental evaluation.

The preceding lab tests also indicated performance gains of CDOXplorer compared to early versions of the algorithm that did not already implement the adaptivity and hybridity characteristics (cf. Section IV-E). A more detailed investigation and quantification of the performance implications, that are induced by integrating the adaptivity and hybridity characteristics, is a worthwhile area for future analyses.

The runs for CDOXplorer, SI-RS, SY-RS, and SI-AN were each repeated 40 times for single as well as for multi cloud scenarios. Thus, we conducted 320 optimization runs with 960,000 simulations in total. We therefore obtained 480,000 simulations for each of the scenarios. These simulations were used to approximate two pareto-optimal fronts that were set to OPF_{Best} for evaluating the respective scenarios. The first of these pareto-optimal fronts (OPF_{Best} for the single cloud scenario) that results from 480,000 simulations is shown in Fig. 15.

B. Experimental Setting

For incorporating public clouds, we measured the MIPIPS values (see Section III-A) for 12 VM instance types of Amazon EC2, and five of Microsoft Windows Azure, by using locations in Europe for both. Five VM instance types of our private Eucalyptus cloud were also benchmarked. For analyzing single and multi cloud scenarios, we used the resulting cloud profiles in the two corresponding scenarios SC_S and SC_M as follows.

$\mathbf{SC_S}$: Amazon EC2

$\mathbf{SC_M}$: Amazon EC2, Microsoft Windows Azure, Eucalyptus

We extracted a KDM architectural model from the open source ERP system Apache OfBiz 10.04.[4] OfBiz is a web-based application, available as open source software, and follows a three-tier architecture. The backend modules are built using Java and Java EE technologies. The presentation tier can employ several different technologies, such as Java Server Pages (JSP). Data is stored in the data layer using a relational database system. Apache OFBiz is built of components that use a common data model and implement specific business functionality. For example, Apache OFBiz provides components for order management, manufacturing management, warehouse management, promotion and pricing management, and customer management.

We deployed Apache OFBiz on a machine of our local cluster and described the deployment in a status quo deployment model. Customers that browse the webstore and put articles in their shopping carts were emulated by producing workload according to a typical day/night usage pattern. More customers visited the webstore in the evening instead in the morning hours and the traffic largely reduced at night. The measured response times and the MIPIPS value of our hardware were then used in a workload profile for generating CDOs and driving the CDOSim simulations. The SLA violation threshold was set to 2s. For Eucalyptus, we defined a synthetic cost model where the prices for VMs follow the capabilities of our VM instance types.

Table II lists the configuration of CDOXplorer that is used within the scope of this evaluation for SC_M.

C. RQ1: Feasibility

This research question evaluates the feasibility and applicability of CDOXplorer. As a basic requirement,

[4]http://ofbiz.apache.org/

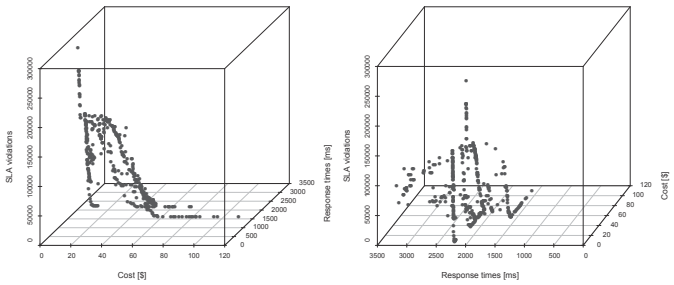


Fig. 15. OPF_{Best} for the single cloud scenario

CDOXplorer has to reliably provide well-suited results. The evaluation of this criteria is of particular importance because of the following two reasons. (1) The simulations used for our simulation-based algorithm are computationally expensive. Hence, we strictly limited the number of generations and population size. This could affect CDOXplorer's capability for producing well-suited approximations of pareto-optimal fronts. (2) The non-determinism used in CDOXplorer, for example, regarding the selection of crossover points, could possibly lead to considerable variations among optimization runs. We approach RQ1 by computing two further metrics `M1` and `M2`.

`M1` analyzes the quality of the results produced by CDOXplorer. We were interested in the degree the hypervolumes of OPF_{Best} can be approximated by CDOXplorer for SC_S and SC_M. For SC_S and SC_M, the HV of OPF_{Best} was 0.462 and 0.573, respectively, whereas CDOXplorer achieved 0.448 and 0.565. Thus, the actual quality of the found pareto optima were sufficiently well-suited for our needs, as these results turn into 96.96% and 98.56% approximation of OPF_{Best} for the single and multi cloud scenario SC_S and SC_M, respectively.

The second metric `M2` calculates the coefficient of variation (CV) that gives information about the relative dispersion regarding a sample's mean value μ. As our performance metrics IGD and HV deliver results in artificial and incomparable units, we convert the standard deviation σ for SC_S and SC_M in combination with IGD and HV to relative and therefore comparable values. These are denoted as CV_S for our single and CV_M for our multi cloud scenario. In general, CV is computed by $CV = \frac{\sigma}{\mu}$. The IGD results for SC_S and SC_M vary in a band of 7.77% and 15.84% around μ, and of 0.46% and 0.32% around μ for HV, respectively. Hence, IGD results in the single cloud scenario are up to 3.85% lower or higher than μ, for instance. Thus, the results indicate that CDOXplorer can reliably find well-suited solutions. However, the value for IGD increases for a higher

TABLE II
CONFIGURATION OF CDOXPLORER FOR THE EXPERIMENTAL EVALUATION OF SC_M. THE VARIABLES REFER TO EQUATION 8.

Variable	Value	Comment
c	3	Three cloud environments are investigated (Amazon EC2, Eucalyptus cluster, and Microsoft Windows Azure (VM role)).
S	$\{S_1\}$	There exists one service, as Apache OFBiz is deployed to a single node in the status quo deployment model ($\|S\| = 1$).
v	2	The maximum number of VM instances that are allowed to start initially in a simulation of CDOSim.
t_1	12	The number of VM instance types defined in the cloud profile of Amazon EC2.
t_2	5	The number of VM instance types defined in the cloud profile of the Eucalyptus cluster.
t_3	5	The number of VM instance types defined in the cloud profile of Microsoft Windows Azure (VM role).
$a_1 - a_3$	3	Maximum of three concurrent node configurations for Amazon EC2, Eucalyptus cluster, and Microsoft Windows Azure (VM role).

number of cloud profiles. In our future work, we will investigate whether this constitutes an actual trend.

D. RQ2: Competitiveness

This research question addresses the competitiveness with other state-of-the-art search approaches by comparing CDOXplorer with SI-RS, SY-RS, and SI-AN. Tables III and IV list the results for the metrics IGD and HV for the single cloud scenario SC_S, respectively. Tables V and VI show these results for SC_M. Tables III-VI list the mean, standard deviation, median, min, and max values of 40 complete, repeated optimization runs for each combination of metric, scenario, and search method. Bigger values are better for HV but worse for IGD. All best mean and median values are set in bold. As can be seen, CDOXplorer outperforms all other search methods in SC_S and also in SC_M.

TABLE III
SEARCH OVER A SINGLE CLOUD PROFILE (SC_S)
(IN INVERTED GENERATIONAL DISTANCE, IGD)

Metric	CDOXplorer	SI-RS	SY-RS	SI-AN
Mean	**2.70E-02**	3.67E-02	4.11E-02	3.28E-02
SD	2.10E-03	2.13E-03	3.61E-03	2.85E-03
Median	**2.72E-02**	3.65E-02	4.21E-02	3.20E-02
Min (best)	2.16E-02	3.34E-02	3.40E-02	2.76E-02
Max (worst)	3.03E-02	4.07E-02	4.83E-02	3.95E-02

We use the Mann-Whitney non-parametric test to evaluate statistical significance. The null hypothesis H_0 states that the results from CDOXplorer cannot be distinguished from those of SI-RS, SY-RS, and SI-AN. Using the Mann-Whitney test and a Bonferroni correction ($\alpha_1 = 0.016$) for multiple comparisons, H_0 is rejected with significance level (α) 0.05 for all combinations of metrics with SC_S and SC_M. Thus, we can quantify the degree CDOXplorer performs better and compare the medians of SI-RS, SY-RS, and SI-AN to those of CDOXplorer. Fig. 16 shows the corresponding fractions with regard to the medians, e.g., for the SC_M scenario, the median for IGD is over 60% lower than that of SY-RS.

Besides significance, we also analyze the effect size when using CDOXplorer in contrast to the other techniques. As proposed by Arcuri and Briand [58] for assessing randomized algorithms in software engineering research, we use the effect size measure $\hat{A}_{12}$ by Vargha and Delaney [59]. For example, consider a comparison of two algorithms A and B. A would deliver 70% of the time better results than B if $\hat{A}_{12} = 0.7$, whereas A and B would be equivalent if $\hat{A}_{12} = 0.5$. As $\hat{A}_{12}$ assumes higher values produced by algorithms to be better, we report $\hat{A}_{12}$ for HV and $(1 - \hat{A}_{12})$ for IGD, as higher values are—as described before—better for HV, but worse in the case of IGD. Tables VII and VIII show the corresponding effect sizes when comparing CDOXplorer to SI-RS, SY-RS, and SI-AN for the different scenarios SC_S and SC_M, respectively. The results for SC_S (overall) and also for SI-RS and SY-RS in the context of SC_M are very good. For example, regarding the single cloud scenario SC_S and IGD, CDOXplorer delivers better results than SI-AN in approximately 97% of the time. Regarding SI-AN and SC_M, the results are still well-suited, but lower than in the case of SC_S. We will analyze in our future work if this constitutes an actual trend when incorporating more cloud profiles in the optimization process.

TABLE IV
SEARCH OVER A SINGLE CLOUD PROFILE (SC_S)
(IN HYPERVOLUME, HV)

Metric	CDOXplorer	SI-RS	SY-RS	SI-AN
Mean	**4.48E-01**	4.41E-01	4.41E-01	4.44E-01
SD	2.08E-03	1.96E-03	2.89E-03	2.09E-03
Median	**4.48E-01**	4.40E-01	4.41E-01	4.44E-01
Min (worst)	4.44E-01	4.36E-01	4.35E-01	4.40E-01
Max (best)	4.54E-01	4.46E-01	4.46E-01	4.48E-01

TABLE V
SEARCH OVER THREE CLOUD PROFILES (SC_M)
(IN INVERTED GENERATIONAL DISTANCE, IGD)

Metric	CDOXplorer	SI-RS	SY-RS	SI-AN
Mean	**3.08E-02**	7.18E-02	8.03E-02	3.37E-02
SD	4.88E-03	2.78E-03	4.41E-03	4.50E-03
Median	**3.12E-02**	7.17E-02	7.90E-02	3.38E-02
Min (best)	2.13E-02	6.58E-02	7.23E-02	2.52E-02
Max (worst)	4.16E-02	7.76E-02	8.88E-02	4.67E-02

In addition to the usage of metrics IGD and HV, we are also interested in how the optimization approaches compare in terms of the evolution of their raw objective values over time. Considering the results observed before, we focus on the two nature-inspired algorithms CDOXplorer and SI-AN and compare these approaches for both scenarios SC_S and SC_M as shown in Fig. 17. For each of the four optimization runs, we report four (intermediate) pareto-optimal fronts that were found by the algorithms after evaluating 750, 1,500, 2,250, and 3,000 individuals, which corresponds to CDOXplorer's generations 15, 30, 45, and 60, respectively. Two patterns can be observed for both SC_S and SC_M: (1) At the beginning to the middle of the optimization runs (columns 1 and 2 in Fig. 17), the intermediate pareto-optimal fronts of SI-AN include considerably fewer non-dominated individuals and many higher (worse) raw objective values than CDOXplorer. This also corresponds to the general functioning of simulated annealing algorithms, that exhibit a higher temperature (i.e., a greater instability) in early phases. (2) While SI-AN approaches the results of CDOXplorer after 2,250 and 3,000 evaluated individuals (columns 3 and 4 in Fig. 17), SI-AN's finally found pareto-optimal fronts (PF_{Known}) are still more sparse. Hence, SI-AN's approximation of OPF_{Best} via these pareto-optimal fronts do not reach CDOXplorer's (diversity) quality. This also corresponds to the measured values of the performance metrics IGD and HV.

TABLE VI
SEARCH OVER THREE CLOUD PROFILES (SC_M)
(IN HYPERVOLUME, HV)

Metric	CDOXplorer	SI-RS	SY-RS	SI-AN
Mean	**5.65E-01**	5.20E-01	5.18E-01	5.63E-01
SD	1.82E-03	1.95E-03	2.43E-03	1.68E-03
Median	**5.65E-01**	5.20E-01	5.17E-01	5.63E-01
Min (worst)	5.61E-01	5.16E-01	5.13E-01	5.61E-01
Max (best)	5.70E-01	5.25E-01	5.25E-01	5.68E-01

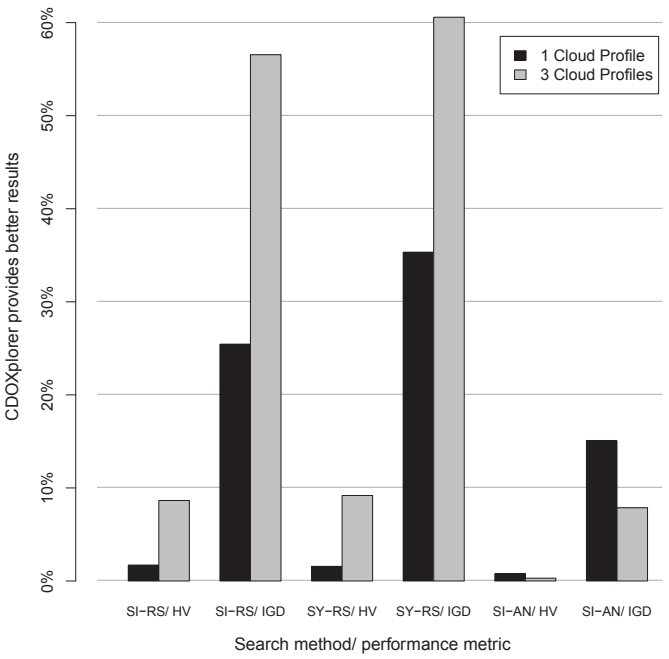


Fig. 16. CDOXplorer advantage relative to other approaches.

E. Threats to Validity

There are several issues that could be a threat to validity. First, we only consider three cloud profiles, as their construction is not trivial and the VM instance types of additional cloud providers need to be benchmarked. This involves potential expenses. However,

TABLE VII
EFFECT SIZE – SEARCH OVER A SINGLE CLOUD PROFILE (SC_S) (COMPARISON OF CDOXPLORER WITH OTHER ALGORITHMS USING $\hat{A}_{12}$ FOR HV AND (1-$\hat{A}_{12}$) FOR IGD)

Metric	SI-RS	SY-RS	SI-AN
IGD	1.00E+00	1.00E+00	9.69E-01
HV	9.96E-01	9.85E-01	8.99E-01

TABLE VIII
EFFECT SIZE – SEARCH OVER THREE CLOUD PROFILES (SC_M) (COMPARISON OF CDOXPLORER WITH OTHER ALGORITHMS USING $\hat{A}_{12}$ FOR HV AND (1-$\hat{A}_{12}$) FOR IGD)

Metric	SI-RS	SY-RS	SI-AN
IGD	1.00E+00	1.00E+00	6.67E-01
HV	1.00E+00	1.00E+00	6.92E-01

scalability seems to be a worthwhile area for further analyses.

Restrictions were also made concerning the workload profile and the studied enterprise software. In each case, just one sample was used. This is due to the fact that further optimization runs imply even more time-consuming simulations. Therefore, we strive for using representative instances. Day/night usage patterns with higher and lower demand are frequently found for enterprise systems. Furthermore, Apache OfBiz is very popular and widespread.

There could also exist other optimization methods that provide better results than SI-RS, SY-RS, and SI-AN we used for evaluation. Though, tailoring optimization methods for our context is time-consuming and not straight forward. Quite similarly, there could exist better ways to tailor SI-AN instead of reusing the mutation sub operators. Especially because SI-AN partially comes near to CDOXplorer's results.

Further threats to validity arise from the synthetic cost model for our private Eucalyptus cloud and that SLAs are usually defined in terms of percentile ranges when considering response times. However, aligning the VM prices to the capabilities of the VM instance types is omnipresent with respect to public cloud environments. Altering the absolute threshold into percentiles for defining the SLA objective can be easily done and will be addressed in the future work.

Additional threats to validity come from the discretization of the search space. The discretization of parameters to be optimized is a standard approach when using genetic algorithms [39]. Nevertheless, there might be subtle discretization errors as the chosen resolution, shown in Table I, might not be high enough. The corresponding gene ranges were determined using domain knowledge. For example, as VM instances often comprise comprehensive software stacks including, among others, operating systems and application frameworks, it usually lasts up to a few minutes to launch them [60]. Hence, it would not be reasonable, for example, for genes G5 and S5 (condition time periods for starting/stopping of VM instances, see Table I) to cover ranges in the order of seconds. Nonetheless, investigating the tradeoffs regarding adjusted resolutions and affected computational costs is a worthwhile subject for further analyses.

VI. RELATED WORK

This section discusses related work concerning the optimization of deployment architectures and reconfiguration rules. Here, solely approaches that tackle related problems from a user perspective are considered. Cloud users want to deploy software to the cloud under given constraints. Especially, a cloud environment's internal structure is transparent to cloud users. As there exists a large body of work in this regard, we limit the description to selected approaches.

Section VI-A discusses related work that stems from deployment optimization in non-clouds scenarios, whereas the next Sections VI-B and VI-C cover approaches that specifically target a cloud context: Approaches using non-evolutionary optimization techniques are described in Section VI-B. Approaches that do employ evolutionary optimization techniques follow in Section VI-C. Runtime cloud deployment adaptation is discussed in Section VI-D.

A. Deployment Optimization in Non-Cloud Scenarios

For improving the deployment architecture of distributed systems regarding arbitrary QoS properties, Malek et al. [61] propose an extensible framework and visual modeling and analysis environment. The framework incorporates continuous system monitoring and changing the deployment architecture at runtime by actually executing redeployment operations. A further mode allows for offline simulation. Arbitrary deployment constraints and QoS properties can be formally defined by manually specifying utility functions. The approach provides four predefined deployment improvement algorithms, among those is

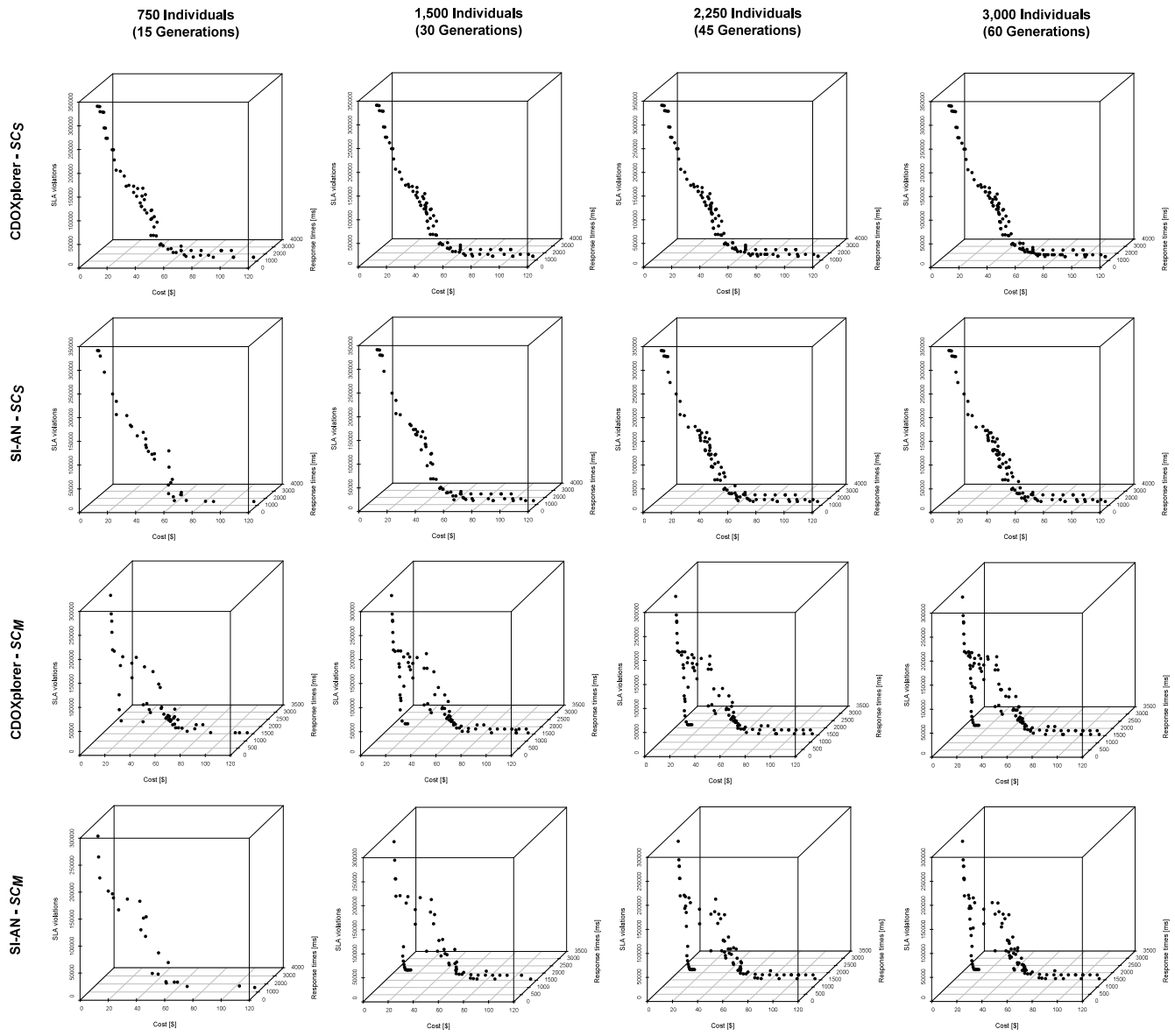


Fig. 17. Evolution of objective values over time for optimization runs of CDOXplorer (first row: SC_S, third row: SC_M) and simulated annealing (second row: SC_S, fourth row: SC_M). Each optimization run uses 3,000 individuals. A column includes the found pareto-optimal fronts after a specific number of evaluated individuals (750, 1,500, 2,250, and 3,000 for columns 1-4, respectively). For CDOXplorer, the corresponding number of generations are stated in brackets. The final optimization results (PF_{Known}) are shown in column 4.

a genetic algorithm. The utility functions are used as fitness functions, whereas CDOXplorer uses simulation runs to obtain fitness values. Our approach does not assume existing knowledge regarding specific utility functions. It instead copes with QoS characteristics and their interdependencies by employing an event-based simulation that successively evolves the CDO elements' state. However, utilizing the event-based simulations of CDOXplorer is more costly than solving the utility functions of an analytical model, as is done by Malek et al. [61]. Moreover, our approach allows integrating reverse-engineered code and component models and the simulation of real workload—upon deployment option candidates—that was monitored at the status quo deployment. While the approach by Malek et al. [61] is very flexible and tailorable to many deployment scenarios, our approach focuses on the IaaS cloud context. This enables CDOXplorer's reconfiguration rules to not only cope with the re-location of components among fixed hardware hosts. Instead, our approach can also consider the cloud's elasticity and explore changes to the infrastructure itself by simulating, for example, the addition and termination of VMs, using different VM instance types, and varying corresponding change conditions.

Martens et al. [62] focus on component-based systems and at finding optimized software and deploy-

ment architectures concerning performance, reliability, and costs. Similar to our approach, a genetic algorithm is used and simulations are, partially, employed for assessing candidate solutions. Their software PerOpteryx provides tool support and explores four degrees of freedom, e.g., processor speeds. Dynamic resource scaling is not supported, but further degrees of freedom can be added. In contrast, CDOXplorer currently explores 17 fixed degrees of freedom (the number of node configurations and 16 genes) and optimizes runtime reconfiguration rules.

A hybrid approach that considers the consolidation of multi-tier applications to virtual machines is presented by Jung et al. [63]. Applications are modeled and optimized offline to generate well-suited system configurations. These are transformed to adaptation policies that can be consumed online by rule engines. Compared with CDOXplorer, the approach by Jung et al. [63] only allows to define a single resource type and homogeneous resource instances instead of considering distinct VM instance types that can also be used in parallel [64].

Zhang et al. [65] propose a QoS-aware approach for finding the optimal number of machines for deploying services in the context of service oriented architectures (SOAs). A greedy algorithm is used that maximizes the throughput. Unlike in our approach, dynamic resource scaling is not supported.

B. Non-Evolutionary Cloud Deployment Optimization

As with our approach CloudMIG, Wu et al. [66] consider resource usage optimization for SaaS providers that build upon leased VMs. Wu et al. [66] assume that a maximum service utilization is defined per customer in SLA agreements. SLA violations and infrastructure costs are minimized by two custom-made algorithms that are evaluated with the tool CloudSim. Our simulator CDOSim also builds on CloudSim. In contrast to CDOXplorer, Wu et al. [66] do not consider distributed applications, runtime reconfiguration, and arbitrary workload.

Trummer et al. [67] contribute an algorithm for computing an application's cost-optimal deployment architecture for IaaS-based clouds. This single-objective optimization is described as a constraint optimization problem and is tackled with an existing constraint solver. As opposed to this, we use multi-objective optimization and support dynamic resource scaling. Nevertheless, the application templates used by Trummer et al. [67] serve the same purpose as our status quo deployment models.

San Aniceto et al. [68] also use a custom-made single-objective optimization algorithm for reducing the costs of leased VMs. Besides on-demand instances—that can be started and stopped at any time—it also considers reserved instances. Those are frequently offered by cloud providers for a single payment per time period. In turn, a discount is given in the hour rates. The algorithm aims at finding the best suited combination of on-demand and reserved instances based on historical workload data. However, it does not support dynamic reconfiguration.

Dynamic scaling is, in contrast, supported by the approach of Mao et al. [69]. Here, integer programming problems are solved for reducing costs or maximizing the performance of scientific computing jobs. Scaling policies are derived at runtime by learning from previous job executions. Compared with our approach, we target enterprise software and use simulations for assessing different CDO candidates before actually deploying an application to a specific cloud environment.

C. Evolutionary Cloud Deployment Optimization

Most related work that employs evolutionary optimization techniques in cloud deployment scenarios takes a cloud provider perspective or requires corresponding knowledge regarding the internal structure of a cloud environment (e.g., [70], [71]). In contrast, Wada et al. [72] follow, in common with CloudMIG, a cloud user perspective and contribute the genetic algorithm E^3-R that explores deployment configurations for optimizing services' QoS attributes. E^3-R can also reduce redundant QoS objectives and estimate the performance of configurations using queueing theory and historic mean arrival rates. However, E^3-R does not support varying workload and dynamic resource scaling, and it regards services as black boxes, whereas our approach simulates reverse-engineered and transformed architectural models and monitored workload.

A particle swarm optimization-based heuristic is introduced by Pandey et al. [73]. The single-objective optimization algorithm maps scientific tasks to cloud resources for minimizing costs. In comparison, we target optimal CDOs for enterprise software, formulate the search as a multi-objective optimization problem, and facilitate dynamic resource scaling.

D. Runtime Cloud Deployment Adaptation

Runtime cloud deployment self-adaptation [74] requires monitoring and observation of running systems

[75]. Opposed to CloudMIG which adresses design-time adaptation, for instance the approaches SLAstic [29], [30], SAM [76] for TeeTime [77], and CapMan [78] for ExplorViz [79]–[83] address runtime adaptation.

VII. Conclusions

We present our simulation-based genetic algorithm CDOXplorer that optimizes cloud deployment options (CDOs) for supporting the migration of enterprise software to Infrastructure as a Service (IaaS)-based cloud environments. A CDO determines which cloud environment, number of virtual machines (VMs), VM instance types, mapping of existing components to VMs, and runtime reconfiguration rules should be used. Runtime reconfiguration rules enable exploiting the cloud's elasticity and describe when—i.e., at which overall VM utilization threshold—to start or stop how much VM instances of what VM instance type with which deployed software components. The runtime reconfiguration rules allow for the definition of both horizontal and vertical scaling strategies.

Automatically finding near-optimal solutions is approached by using techniques of the search-based software engineering field. CDOs become optimized in terms of response times, costs, and number of SLA violations. For example, the best-suited cloud environment and VM instance types for an existing software system and particular usage patterns have to be found. CDOXplorer uses our tool CDOSim to simulate CDOs. A simulation result represents a fitness value of our genetic algorithm.

As with all simulation-based optimization approaches, the evaluation of the fitness function (the simulation) is computationally very expensive. As a CDO simulation can last from minutes to hours, we rigorously restricted the genetic algorithm's population size and number of generations and calibrated these parameters to 50 and 60, respectively. CDOXplorer is an adaptive and hybrid genetic algorithm. It uses adaptive mutation and crossover rates to increase the convergence speed and also employs a combination with a local search technique to further improve its search results.

The popular public clouds Amazon EC2 and Microsoft Windows Azure, as well as our private Eucalyptus cloud, were used for extensive experiments. We compared CDOXplorer with three state-of-the-art search and optimization methods using, among others, the two standard performance metrics inverted generational distance (IGD) and hypervolume indicator (HV). Our evaluation showed that (1) CDOXplorer can reliably find well-suited results despite the necessary rigorous restrictions of population size and number of generations and that (2) it can find CDOs that are up to 60% better than those of the other approaches.

Our experiment code and data and an implementation of CDOXplorer are available online as open source software such that interested researchers may repeat or extend our work.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[2] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing – the business perspective," *Decision Support Systems*, vol. 51, no. 1, pp. 176–189, 2011.

[3] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch, "How to adapt applications for the Cloud environment," *Computing*, vol. 95, no. 6, pp. 493–535, 2013.

[4] P. Jamshidi, A. Ahmad, and C. Pahl, "Cloud Migration Research: A Systematic Review," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 142–157, 2013.

[5] H. Knoche and W. Hasselbring, "Using microservices for legacy software modernization," *IEEE Software*, vol. 35, no. 3, pp. 44–49, May 2018.

[6] W. Hasselbring, "Software architecture: Past, present, future," in *The Essence of Software Engineering*, V. Gruhn and R. Striemer, Eds. Cham: Springer International Publishing, 2018, pp. 169–184. [Online]. Available: https://doi.org/10.1007/978-3-319-73897-0_10

[7] W. Hasselbring and G. Steinacker, "Microservice architectures for scalability, agility and reliability in e-commerce," in *Proceedings 2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. Gothenburg, Sweden: IEEE, Apr. 2017, pp. 243–246.

[8] W. Hasselbring, "Microservices for scalability: Keynote talk abstract," in *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (ICPE 2016)*. New York, NY, USA: ACM, 2016, pp. 133–134.

[9] R. Prodan and S. Ostermann, "A survey and taxonomy of infrastructure as a service and web hosting cloud providers," in *10th IEEE/ACM International Conference on Grid Computing, 2009*, Oct. 2009.

[10] M. Kuperberg, N. Herbst, J. v. Kistowski, and R. Reussner, "Defining and Quantifying Elasticity of Resources in Cloud Computing and Scalable Platforms," Karlsruhe Reports in Informatics 2011, 16, 2011.

[11] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. M. Vaquero, "Service specification in cloud environments based on extensions to open standards," in *Proc. of the Fourth International ICST Conference on Communication System Software and Middleware*. ACM, 2009.

[12] J. Grundy, G. Kaefer, J. Keong, and A. Liu, "Guest Editors' Introduction: Software Engineering for the Cloud," *IEEE Software*, vol. 29, 2012.

[13] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for QoS-aware service composition based on genetic algorithms," in *Proc. of the Conf. on Genetic and Evol. Computation*. ACM, 2005.

[14] M. Harman, K. Lakhotia, J. Singer, D. R. White, and S. Yoo, "Cloud engineering is search based software engineering too," *Journal of Systems and Software*, vol. 86, no. 9, pp. 2225–2241, 2013.

[15] M. Harman, "Software Engineering Meets Evolutionary Computation," *Computer*, vol. 44, no. 10, pp. 31–39, 2011.

[16] F. Fittkau, S. Frey, and W. Hasselbring, "CDOSim: Simulating Cloud Deployment Options for Software Migration Support," in *Proceedings of the 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2012)*. IEEE Computer Society, 2012, pp. 37–46.

[17] S. Frey, W. Hasselbring, and B. Schnoor, "Automatic conformance checking for migrating software systems to cloud infrastructures and platforms," *Journal of Software: Evolution and Process*, vol. 25, no. 10, pp. 1089–1115, Oct. 2013, doi: 10.1002/smr.582.

[18] S. Frey and W. Hasselbring, "The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications," *Int'l Journal on Advances in Software*, vol. 4, no. 3 and 4, 2011.

[19] ——, "Model-based migration of legacy software systems to scalable and resource-efficient cloud-based applications: The CloudMIG approach," in *Proc- First International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2010)*, Lisbon, Portugal, Nov. 2010, pp. 155–158.

[20] ——, "Model-based migration of legacy software systems into the cloud: The CloudMIG approach," *Softwaretechnik-Trends*, vol. 30, no. 2, pp. 84–85, 2010, (Proc. WSR 2010).

[21] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," Sep. 2011, NIST Special Publication 800-145.

[22] A. Law and M. McComas, "Simulation-based optimization," in *Proc. of the Winter Simulation Conference, 2000*, vol. 1, 2000, pp. 46–49.

[23] S. Frey, F. Fittkau, and W. Hasselbring, "Search-Based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud," in *Proceedings of the 2013 International Conference on Software Engineering (ICSE 2013)*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 512–521.

[24] S. Frey, "Conformance Checking and Simulation-based Evolutionary Optimization for Deployment and Reconfiguration of Software in the Cloud," PhD Thesis, Kiel University, 2014. [Online]. Available: http://eprints.uni-kiel.de/23862/

[25] R. Pérez-Castillo, I. G.-R. de Guzmán, and M. Piattini, "Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems," *Computer Standards & Interfaces*, vol. 33, no. 6, 2011.

[26] H. Brunelière, J. Cabot, and G. D. F. Madiot, "Modisco: a model driven reverse engineering framework," *Information and Software Technology*, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2014.04.007

[27] A. van Hoorn, S. Frey, W. Goerigk, W. Hasselbring, H. Knoche, S. Köster, H. Krause, M. Porembski, T. Stahl, M. Steinkamp, and N. Wittmüss, "Dynamod project: Dynamic analysis for model-driven software modernization," in *Proceedings of the 1st International Workshop on Model-Driven Software Migration (MDSM 2011)*, vol. 708. CEUR, 2011, pp. 12–13.

[28] A. van Hoorn, J. Waller, and W. Hasselbring, "Kieker: A framework for application performance monitoring and dynamic software analysis," in *Proc. of the 3rd ACM/SPEC Int'l Conference on Performance Engineering (ICPE 2012)*. ACM, Apr. 2012, pp. 247–248.

[29] R. von Massow, A. van Hoorn, and W. Hasselbring, "Performance simulation of runtime reconfigurable component-based software architectures," in *Software Architecture (Proceedings ECSA 2011)*, ser. Lecture Notes in Computer Science, I. Crnkovic, V. Gruhn, and M. Book, Eds., vol. 6903. Springer-Verlag, 2011, pp. 43–58.

[30] A. van Hoorn, M. Rohr, I. A. Gul, and W. Hasselbring, "An adaptation framework enabling resource-efficient operation of software systems," in *Proc. of the Warm Up Workshop (WUP 2009) for ACM/IEEE ICSE 2010*, N. Medvidovic and T. Tamai, Eds. ACM, Apr. 2009, pp. 37–40.

[31] J. D. Cryer and K.-S. Chan, *Time Series Analysis: With Applications in R (Springer Texts in Statistics)*. Springer, Nov. 2010.

[32] S. Dustdar, Y. Guo, R. Han, B. Satzger, and H.-L. Truong, "Programming Directives for Elastic Computing," *IEEE Internet Computing*, vol. 16, no. 6, pp. 72–77, Nov. 2012.

[33] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182 –197, Apr. 2002.

[34] S. Frey and W. Hasselbring, "An extensible architecture for detecting violations of a cloud environment's constraints during legacy software system migration," in *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR 2011)*, T. Mens, Y. Kanellopoulos, and A. Winter, Eds. IEEE Computer Society, Mar. 2011, pp. 269–278.

[35] B. Craenen, A. Eiben, and E. Marchiori, "How to Handle Constraints with Evolutionary Algorithms," in *The Practical Handbook of Genetic Algorithms*. Chapman and Hall/CRC, Dec. 2000.

[36] M. Srinivas and L. Patnaik, "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, no. 4, pp. 656–667, Apr. 1994.

[37] Y. Yun and M. Gen, "Performance Analysis of Adaptive Genetic Algorithms with Fuzzy Logic and Heuristics," *Fuzzy Optimization and Decision Making*, vol. 2, pp. 161–175, 2003.

[38] N. L. Law and K. Y. Szeto, "Adaptive Genetic Algorithm with Mutation and Crossover Matrices," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, ser. IJCAI'07. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, pp. 2330–2333.

[39] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, 1994, 10.1007/BF00175354.

[40] M. Buckley, "Linear Array Synthesis Using a Hybrid Genetic Algorithm," in *Antennas and Propagation Society International Symposium, 1996. AP-S. Digest*, vol. 1, Jul. 1996, pp. 584–587 vol.1.

[41] Y. Nie and W. Deng, "A Hybrid Genetic Learning Algorithm for Pi-Sigma Neural Network and the Analysis of Its Convergence," in *Fourth International Conference on Natural Computation, 2008. ICNC '08.*, vol. 3, Oct. 2008, pp. 19–23.

[42] Z. Man, T. Wei, L. Xiang, and K. Lishan, "Research on Multi-project Scheduling Problem Based on Hybrid Genetic Algorithm," in *International Conference on Computer Science and Software Engineering, 2008*, vol. 1, Dec. 2008, pp. 390–394.

[43] J. Sha and M. Xu, "Applying Hybrid Genetic Algorithm to Constrained Trajectory Optimization," in *International Conference on Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011*, vol. 7, Aug. 2011, pp. 3792 –3795.

[44] H. Ishibuchi and T. Murata, "Local Search Procedures in a Multi-Objective Genetic Local Search Algorithm for Scheduling Problems," in *IEEE International Conference on Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999*, vol. 1, 1999, pp. 665 –670 vol.1.

[45] H. Ishibuchi, T. Yoshida, and T. Murata, "Balance Between Genetic Search and Local Search in Memetic Algorithms for Multiobjective Permutation Flowshop Scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 204–223, Apr. 2003.

[46] J. Bhuvana and C. Aravindan, "Preferential local search with adaptive weights in Evolutionary Algorithms for Multiobjective Optimization Problems," in *International Conference of Soft Computing and Pattern Recognition (SoCPaR), 2011*, Oct. 2011, pp. 358–363.

[47] F. Fittkau, S. Frey, and W. Hasselbring, "Cloud User-Centric Enhancements of the Simulator CloudSim to Improve Cloud Deployment Option Analysis," in *Proc. of the European Conference on Service-Oriented and Cloud Computing (ESOCC)*, ser. Lecture Notes in Computer Science. Bertinoro, Italy: Springer Berlin / Heidelberg, Sep. 2012, vol. 7592, pp. 200–207.

[48] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4J: A Modular Framework for Meta-Heuristic Optimization," in *Proc. of the 13th Annual Conf. on Genetic and Evolutionary Computation*. ACM, 2011.

[49] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, Apr. 2003.

[50] Q. Zhang, A. Zhou, and Y. Jin, "RM-MEDA: A Regularity Model-Based Multiobjective Estimation of Distribution Algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 41–63, Feb. 2008.

[51] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms - a comparative case study," in *Parallel Problem Solving from Nature*, ser. Lecture Notes in Comp. Science. Springer, 1998, vol. 1498.

[52] T. G. Kolda, R. M. Lewis, and V. Torczon, "Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods," *SIAM Review*, vol. 45, pp. 385–482, 2003.

[53] F. Schoen, "Stochastic techniques for global optimization: A survey of recent advances," *Journal of Global Optimization*, vol. 1, 1991.

[54] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, Sep. 2003.

[55] P. K. B. Suman, "A Survey of Simulated Annealing as a Tool for Single and Multiobjective Optimization," *The Journal of the Operational Research Society*, vol. 57, no. 10, pp. 1143–1160, 2006.

[56] P. Serafini, *Multiple Criteria Decision Making: Proceedings of the Tenth International Conference: Expand and Enrich the Domains of Thinking and Application*. New York, NY: Springer New York, 1994, ch. Simulated Annealing for Multi Objective Optimization Problems, pp. 283–292.

[57] A.-T. Nguyen, S. Reiter, and P. Rigo, "A review on simulation-based optimization methods applied to building performance analysis," *Applied Energy*, vol. 113, pp. 1043 – 1058, 2014.

[58] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *33rd International Conference on Software Engineering (ICSE), 2011*, May 2011, pp. 1–10.

[59] A. Vargha and H. D. Delaney, "A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[60] B. Suleiman, S. Sakr, R. Jeffery, and A. Liu, "On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure," *Journal of Internet Services and Applications*, vol. 3, pp. 173–193, 2012.

[61] S. Malek, N. Medvidovic, and M. Mikic-Rakic, "An Extensible Framework for Improving a Distributed Software System's Deployment Architecture," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 73–100, Jan. 2012.

[62] A. Martens, H. Koziolek, S. Becker, and R. Reussner, "Automatically Improve Software Architecture Models for Performance, Reliability, and Cost Using Evolutionary Algorithms," in *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*. ACM, 2010, pp. 105–116.

[63] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu, "Generating Adaptation Policies for Multi-tier Applications in Consolidated Server Environments," in *Int'l Conf. on Autonomic Computing, 2008*, Jun. 2008.

[64] W. Hasselbring, "Programming languages and systems for prototyping concurrent applications," *ACM Computing Surveys*, vol. 31, no. 1, pp. 43–79, Mar. 2000.

[65] C. Zhang, R. Chang, C.-S. Perng, E. So, C. Tang, and T. Tao, "QoS-Aware Optimization of Composite-Service Fulfillment Policy," in *IEEE Int'l Conference on Services Computing, 2007*, Jul. 2007, pp. 11–19.

[66] L. Wu, S. Garg, and R. Buyya, "SLA-Based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments," in *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2011*, May 2011, pp. 195–204.

[67] I. Trummer, F. Leymann, R. Mietzner, and W. Binder, "Cost-optimal outsourcing of applications into the clouds," in *IEEE Second Int'l Conf. on Cloud Computing Technology and Science (CloudCom), 2010*, 2010.

[68] I. San Aniceto, R. Moreno-Vozmediano, R. Montero, and I. Llorente, "Cloud capacity reservation for optimal service deployment," in *Proceedings of the Second International Conference on Cloud Computing, GRIDs, and Virtualization*. IARIA Conference, Sep. 2011, pp. 52–59.

[69] M. Mao, J. Li, and M. Humphrey, "Cloud Auto-scaling with Deadline and Budget Constraints," in *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing (GRID)*, Oct. 2010, pp. 41–48.

[70] M. J. Csorba, H. Meling, and P. E. Heegaard, "Ant system for service deployment in private and public clouds," in *Proc. of the 2nd Workshop on Bio-inspired Algorithms for Distributed Systems*. ACM, 2010.

[71] Z. I. M. Yusoh and M. Tang, "Composite SaaS Placement and Resource Optimization in Cloud Computing Using Evolutionary Algorithms," *IEEE Fifth Int'l Conference on Cloud Computing*, pp. 590–597, 2012.

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

[73] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments," in

Proc. 24th IEEE Int'l Advanced Information Networking and Applications (AINA) Conference, 2010.

[74] M. Rohr, S. Giesecke, W. Hasselbring, M. Hiel, W.-J. van den Heuvel, and H. Weigand, "A classification scheme for self-adaptation research," in *Proceedings of the International Conference on Self-Organization and Autonomous Systems In Computing and Communications (SOAS'2006)*, Sep. 2006, p. 5.

[75] W. Hasselbring, R. Heinrich, R. Jung, A. Metzger, K. Pohl, R. Reussner, and E. Schmieders, "iObserve: integrated observation and modeling techniques to support adaptation and evolution of software systems," Kiel University, Kiel, Germany, Forschungsbericht, Oktober 2013. [Online]. Available: http://eprints.uni-kiel.de/22077/

[76] C. Wulf, C. C. Wiechmann, and W. Hasselbring, "Increasing the throughput of pipe-and-filter architectures by integrating the task farm parallelization pattern," in *Proceedings of the 19th International Symposium on Component-Based Software Engineering (CBSE 2016)*, 2016, pp. 13–22.

[77] C. Wulf, W. Hasselbring, and J. Ohlemacher, "Parallel and generic pipe-and-filter architectures with TeeTime," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, Gothenburg, Sweden, Apr. 2017, pp. 290–293.

[78] F. Fittkau and W. Hasselbring, "Elastic application-level monitoring for large software landscapes in the cloud," in *Service Oriented and Cloud Computing*, ser. Lecture Notes in Computer Science, S. Dustdar, F. Leymann, and M. Villari, Eds., vol. 9306. Springer-Verlag, September 2015, pp. 80–94.

[79] F. Fittkau, A. Krause, and W. Hasselbring, "Software Landscape and Application Visualization for System Comprehension with ExplorViz," *Information and Software Technology*, vol. 87, pp. 259–277, Jul. 2017.

[80] F. Fittkau, S. Roth, and W. Hasselbring, "ExplorViz: Visual runtime behavior analysis of enterprise application landscapes," in *23rd European Conference on Information Systems (ECIS 2015 Completed Research Papers)*. AIS Electronic Library, May 2015, pp. 1–13.

[81] F. Fittkau, A. Krause, and W. Hasselbring, "Hierarchical software landscape visualization for system comprehension: A controlled experiment," in *Proceedings of the 3rd IEEE Working Conference on Software Visualization (VISSOFT 2015)*. IEEE, Sep. 2015, pp. 36–45.

[82] F. Fittkau, S. Finke, W. Hasselbring, and J. Waller, "Comparing trace visualizations for program comprehension through controlled experiments," in *Proceedings of the 23rd IEEE International Conference on Program Comprehension (ICPC 2015)*. IEEE, May 2015, pp. 266–276.

[83] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring, "Live trace visualization for comprehending large software landscapes: The ExplorViz approach," in *1st IEEE International Working Conference on Software Visualization (VISSOFT 2013)*, Sep. 2013, pp. 1–4.